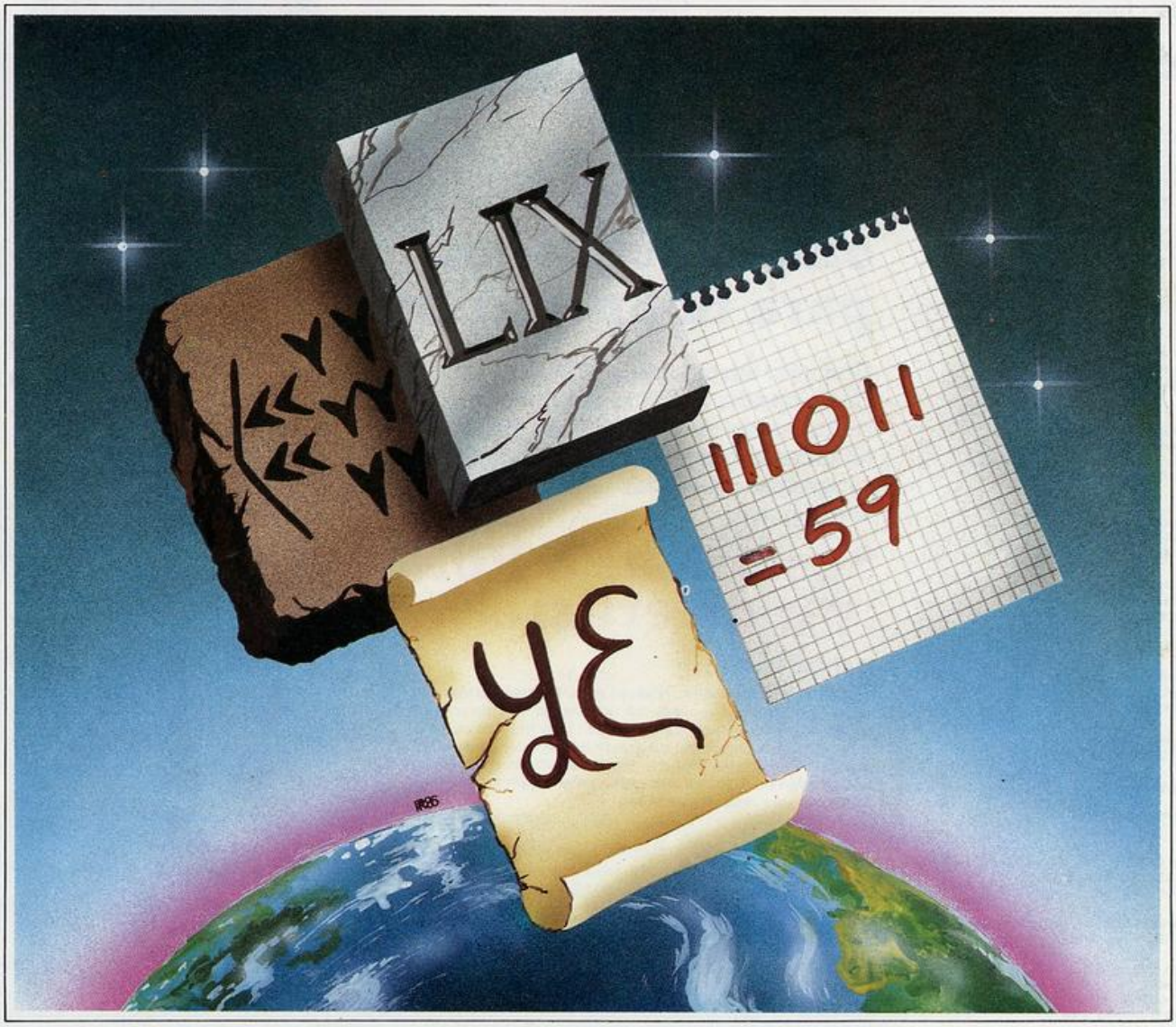
10
150pts.

RUN

Enciclopedia Práctica del Spectrum



Nueva Lente/Ingelek

SENCILLO, ASEQUIBLE, PROFESIONAL

ASI ES EL QL DE SINCLAIR, HECHO PARA NOSOTROS

Para los profesionales que necesitamos un teclado en nuestro idioma, QL nos ofrece, en castellano, su QWERTY standard de 65 teclas móviles.

Para los que deseamos comunicarnos a gran velocidad y capacidad con nuestro ordenador, QL nos presenta su lenguaje SUPER BASIC.

Para los que necesitamos gran margen operativo, ahora disponemos de un ordenador con memoria ROM de 32K que contiene el sistema operativo QDOS, un sistema mono-usuario, multi-tarea y con partición de tiempo.

Para los que deseamos tener perfectamente ordenada nuestra agenda de trabajo, presupuestos, fichas de productos, nuestra correspondencia, estadísticas de venta, archivo... QL viene dotado de cuatro microdrives totalmente interactivados entre sí. QL QUILL de Tratamiento de Textos, QL ARCHIVE Base de Datos, QL ABACUS Hoja Electrónica de Cálculo y el QL EASEL para realización de todo tipo de gráficos.

Para los que nos gustan las cosas bien acabadas, QL se suministra con su fuente de alimentación, cables de conexión y adaptadores de TV, monitor y red local, cuatro programas de software de uso genérico, cuatro cartuchos en blanco para los microdrives y manual de instrucciones en castellano.

Para los que creemos que lo bien hecho puede tener también el mejor precio, ***QL ahora por sólo 125.000 pts.***

Para los que nos gusta siempre ir bien acompañados, Sinclair —el mayor vendedor del mundo en ordenadores personales— e Investrónica, la mayor red de distribución de España, son nuestras mejores Compañías. Nuestra mejor garantía.

En definitiva, para los que queremos ordenarnos y nunca nos habíamos atrevido.

Con QL ya no hay excusas.

DISTRIBUIDOR EXCLUSIVO **investronica**

Tomás Bretón, 60. Telf. (91) 467 82 10. Télex 23399 IYCO E. 28045 Madrid
Camp, 80. Telf. (93) 211 26 58-211 27 54. 08022 Barcelona

EVALUADOR DE CADENAS

LA creación de programas relativamente complejos, requiere del uso de las funciones BASIC. Ya en el último programa de ejemplo propuesto, hemos utilizado la función BASIC **VAL**, aunque sin describir sus características y modo de empleo.

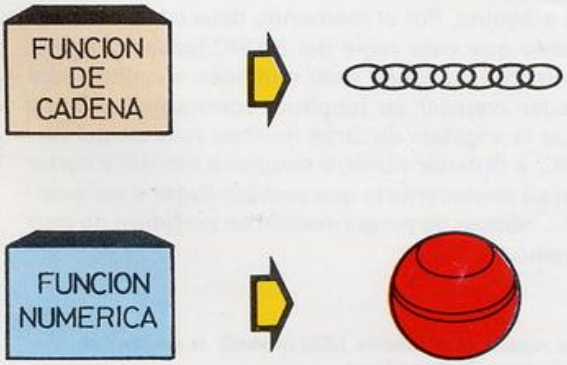
De modo general, podemos decir que las funciones BASIC operan como «traductores», que a partir de un «argumento» que suministramos, producen un valor de salida determinado. De esta forma, la estructura general de cualquier función es:

FUNCION + ARGUMENTO

Como FUNCION, se especifica cualquiera de las múltiples palabras claves de este tipo disponibles en el vocabulario BASIC, y como ARGUMENTO, encerrado entre paréntesis sólo cuando es estrictamente necesario, la expresión a evaluar, que puede ser tan complicada como se quiera; dependiendo de la clase de función elegida, la expresión ARGUMENTO será de tipo numérico o de cadena.

Lógicamente, los argumentos especificados para las diferentes funciones deben ser coherentes con la función escogida, produciéndose, en caso contrario, una detención del programa con el correspondiente mensaje de error. Este será frecuentemente del tipo **C Nonsense in BASIC**, que podemos traducir al Castellano como **C Sinsentido en BASIC** o **C Tontería en BASIC**, si somos más coloquiales y duros con nosotros mismos. En otras muchas ocasiones, los errores de este tipo con funciones, pueden ser ya detectados en la fase de corrección de sintaxis por ese sistema tan eficaz que ya conocemmos: el *syntax checker*.

Las funciones deben operar sobre el tipo de argumento adecuado.



La utilización de una función siempre lleva consigo la presencia de un argumento.

Por los resultados que podemos obtener de ellas, dividimos las funciones en dos grandes grupos, las funciones de cadena y las numéricas. De momento nos ocuparemos de las primeras, que nos permitirán ampliar conocimientos dentro del ámbito del manejo avanzado de cadenas de caracteres, pasando en páginas posteriores al estudio de las funciones de argumento numérico.

Las funciones a que nos referimos en este capítulo son:

STR$, VAL$, VAL, y LEN

De ellas, las dos primeras producen como resultado una cadena de caracteres, mientras que las restantes proporcionan un resultado numérico. Para poder recordar más fácilmente esta circunstancia, basta con advertir que las que producen como resultado una variable de cadena, llevan como sufijo el símbolo dólar **($)**, y las que dan como resultado un valor numérico, no lo llevan, de forma similar a las reglas de construcción de los nombres de variables.

Las funciones pueden admitir como argumento cualquier expresión, siempre que sea del tipo adecuado: de cadena, en este caso. Esto quiere decir que es posible adoptar desde la forma más sencilla, es decir, una constante, hasta nombres de variables o expresiones más complejas por combinación de las anteriores, con inclusión opcional de operadores matemáticos y de otro tipo, o incluyendo otras funciones.

Una parte muy importante de la «inteligencia» de nuestro Spectrum (todo aquello que ya sabe des-

¡!

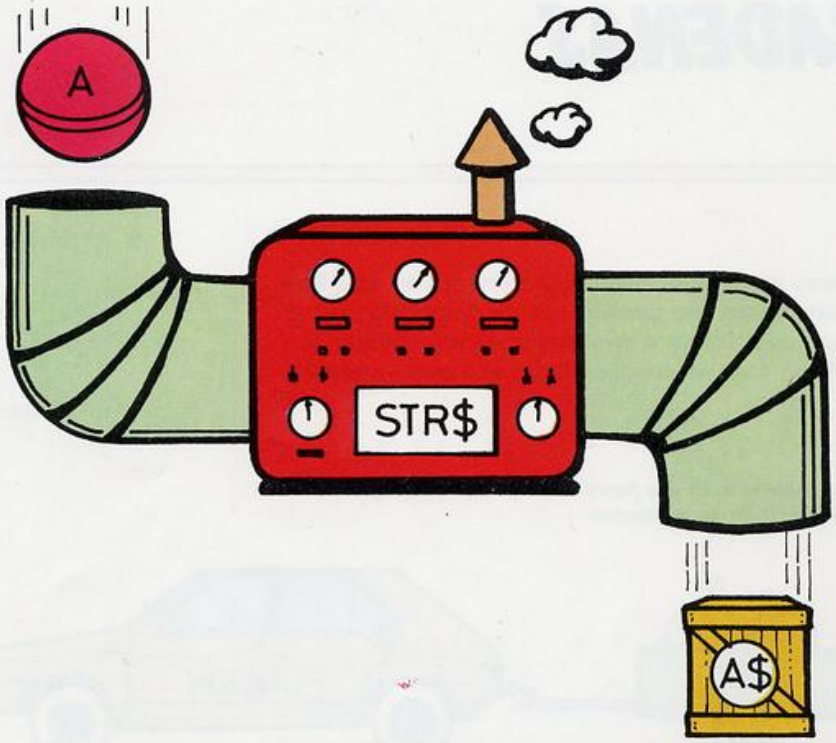
La estructura general de una función es: FUNCION + ARGUMENTO.

*

Los argumentos especificados para las diferentes funciones, deben ser coherentes con la función escogida.

*

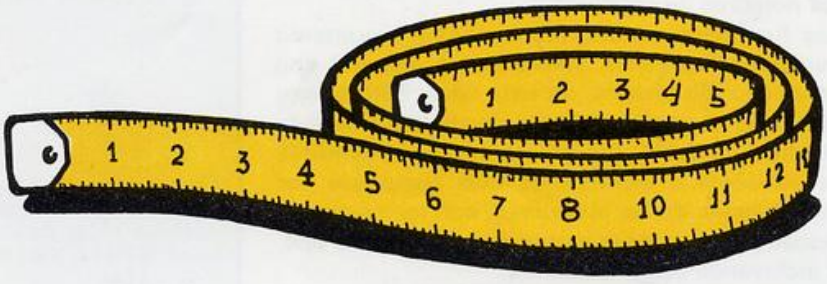
Las funciones destinadas al tratamiento de cadenas en el BASIC son: **STR$, VAL$, VAL** y **LEN**.



La función **STR$** *permite convertir las expresiones numéricas en cadenas.*

de que encendemos la máquina, sin que nosotros se los expliquemos), se emplea en facilitar esta evaluación de expresiones, que nos permite, como ya hemos visto, utilizar expresiones auténticamente complejas en prácticamente cualquier punto del programa, y con casi todas las sentencias. Esto, que a los recién llegados a la informática nos parecerá tan natural, no lo es tanto, y se da la paradoja de que versiones del lenguaje BASIC para equipos con un coste más de diez veces superior al del Spectrum, carecen de esta gran facilidad para la utilización de expresiones.

Para cerrar este inciso, brindamos por tanto una cerrada ovación a nuestro Spectrum, y más concretamente al sistema que nos proporciona una comodidad tan considerable en el manejo de expresiones: EL EVALUADOR. Volvemos ahora al tema que nos ocupa.



LEN

LA FUNCION STR$

Esta función permite obtener la forma de cadena de una expresión numérica. En cierta manera, es como convertir una expresión de un tipo al otro, lo cual, aunque a primera vista pueda parecer poco trascendente, es de la mayor importancia, debido a que algunas funciones, como ya hemos comentado, sólo son accesibles para determinados tipos de variables (o expresiones), y gracias a esta circunstancia lo son para todas ellas, previa conversión de la expresión al tipo adecuado para la función.

La impresión de variables o expresiones numéricas BASIC, puede representar un problema cuando lo que se desea es que aparezcan en la pantalla «columnas» de números. Esto es debido a que la sentencia **PRINT** imprime las variables, tanto numéricas como de cadena de izquierda a derecha, mientras que nosotros estamos más acostumbrados a que las expresiones con texto se justifiquen (columnen) por la izquierda y las numéricas por la derecha.

La solución a este problema es sencilla, y se basa en el empleo de la función **STR$**, en combinación con **LEN** y **TAB** (función que estudiaremos más adelante). El motivo de calcular el equivalente en expresión de cadena de una expresión numérica es, precisamente, poder averiguar su longitud, ya que el BASIC no permite conocer la longitud de una variable o expresión numérica de forma directa, sin su conversión previa a expresión de cadena mediante la función **STR$**.

Este hecho nos puede parecer una considerable incomodidad, pero sin embargo se soporta en una sólida base informática; concretamente, en la forma en que la máquina trata y almacena la información numérica. Actualmente no estamos capacitados para comprender el por qué de esto, pero antes de terminar la lectura de nuestra obra conoceremos estas interioridades del BASIC y de la máquina. Por el momento, debe bastarnos con saber que esta regla del BASIC (pasar obligatoriamente cualquier dato numérico a cadena para poder conocer su longitud) contribuirá, al igual que la «rigidez» de otras muchas normas del BASIC, a ordenar nuestro esquema mental y cortar en su nacimiento lo que podrían llegar a ser grandes «vicios» de programación en un futuro no muy lejano.

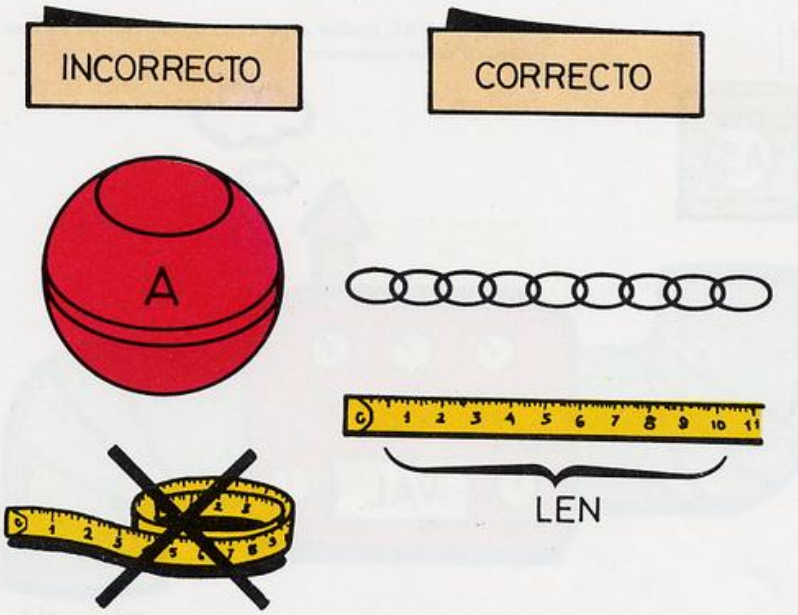
La misión de la función **LEN** *es medir la longitud en caracteres de las cadenas.*

LA FUNCION LEN

A través de la función **LEN,** podemos calcular el número de caracteres contenidos en una variable o expresión de cadena. En caso de que deseemos conocer el número de caracteres que ocupará la impresión de una variable o expresión de tipo numérico, deberemos calcular primero la **STR$** de la expresión (**STR$** es abreviatura de *STRing,* que como ya sabemos, en inglés significa cadena) para, posteriormente, hallar su **LEN** (abreviatura de *LENgth,* que significa longitud). Vamos a ver más claro el uso de **STR$** y **LEN** con algunos ejemplos:

```
10 REM - STR$ Y LEN
20 LET A=-1234
30 LET B=567890
40 LET A$="ABCDE"
50 LET B$="FGHIJKL"
60 CLS
70 PRINT "Cadena:";STR$ A,"Longitud:";LEN STR$ A
80 PRINT "Cadena:";STR$ B,"Longitud:";LEN STR$ B
90 PRINT "Cadena:";STR$ (A+B), "Longitud:";LEN STR$ (A+B)
100 PRINT "Cadena:";A$,"Longitud:";LEN A$
110 PRINT "Cadena:";B$,"Longitud:";LEN B$
120 PRINT "Cadena:";A$+B$,"Longitud:";LEN (A$+B$)
```

La función **LEN** nos permite calcular la longitud (número de caracteres) de que se compone una cadena, o una expresión de cadena como lo son **A$+B$** o **STR$.** De lo dicho se desprende que el astuto *syntax checker* (comprobador de sintaxis), rechazará olímpicamente cualquiera de nuestros intentos por realizar el **STR$** de una expresión de cadena, cosa por otra parte bastante lógica, puesto que no hay que pensárselo mucho para darse cuenta de la inutilidad de convertir en cadena **(STR$)** algo que ya es cadena; habría poco mérito en ello, ¿verdad?

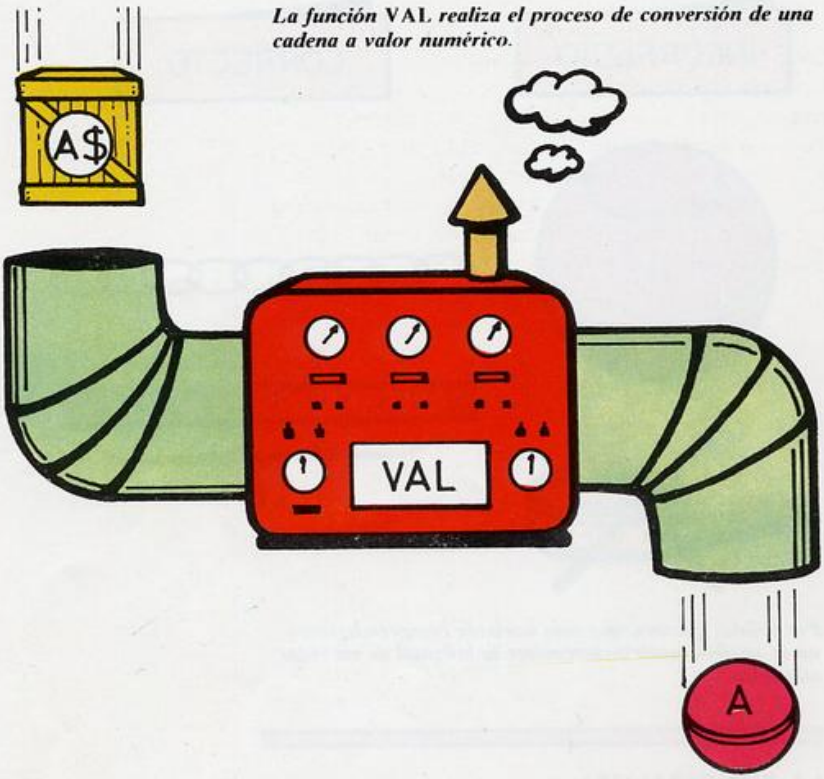


Por sólidas razones, que más adelante comprenderemos, no es posible medir directamente la longitud de un valor numérico.

LA FUNCION VAL

La función **VAL** es, ciertamente, la inversa de la función **STR$,** puesto que calcula el valor numérico de una variable o expresión de cadena. **VAL** es una función de gran potencia, que permite evaluar el contenido de la expresión de cade-

Gracias a la función **VAL**, *podemos emplear nuestro Spectrum como una potente calculadora en la representación de funciones.*



La función **VAL** *realiza el proceso de conversión de una cadena a valor numérico.*

na para calcular su valor numérico. Lo veremos más claro con algunos ejemplos:

```
10 REM - VAL
20 LET A=-1234
30 LET B=567890
40 LET X$="A+B"
50 CLS
60 PRINT X$,VAL X$
```

Hemos comprobado hasta qué punto es capaz la función **VAL** de indagar en el interior de una cadena de caracteres. Todos los elementos que se encuentran dentro de ella, tanto constantes literales como variables y operadores aritméticos, son evaluados para dar como resultado un valor numérico.

A pesar de ello, el caso más común en que se utiliza la expresión **VAL**, es para calcular el valor numérico de una cadena de caracteres, todos ellos numéricos (justo lo inverso a **STR$**).

Sólo con un poquito de imaginación, y habiendo observado la extraordinaria capacidad de análisis de la función **VAL**, debemos intuir que pueden obtenerse los diferentes valores de una función (tabla de valores), que nos permiten dibujar su gráfico. Al fin y al cabo, dada una función matemática cualquiera, por ejemplo *Y=X*X*, hallar la tabla de valores correspondientes para su posterior representación, se limita a ir probando valores de *X* para obtener valores de *Y*; por tanto, ¿qué mejor que la función **VAL** para cumplir ese cometido? Veámosla actuar en el ejemplo propuesto:

```
10 REM - TABLA DE VALORES
20 CLS:PRINT "VALOR DE Y","VALOR DE X"''
30 LET X$="X*X"
40 FOR X=-10 TO 10
50 PRINT X,VAL "X$"
60 NEXT X
```

Esta facilidad en la evaluación de cadenas, que tampoco es frecuente entre otros BASIC, capaces sólo de hallar el **VAL** de constantes, y no de variables o de expresiones, se la debemos agradecer nuevamente al EVALUADOR, puesto que nos proporciona un sistema magnífico para hallar tablas de valores, etc... Sólo tenemos que efectuar por **INPUT** la entrada de la variable de cadena a examinar, y luego **VAL** se ocupará del resto. Así pues, vamos a sustituir la línea 30 del último ejemplo, que asigna a la cadena X$ la función a evaluar, por **30 INPUT "Y= ";X$**, lo que nos va a permitir introducir varias funciones sin necesidad de tener que cambiar cada vez el programa.

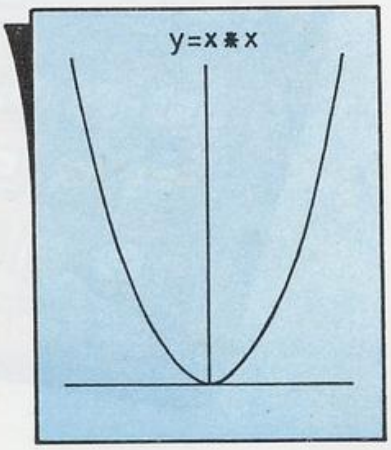
Pero esta facilidad se puede volver en contra nuestra. Si forzamos a calcular el valor numérico de una expresión de cadena en la que está contenido algún carácter no numérico, el Spectrum interpreta que se trata de una variable, y busca en la memoria su valor para reemplazarlo en la expresión. En caso de no encontrarlo, el programa se detiene con el mensaje de error **2 Variable not found** (variable no encontrada).

Esto sucederá en el ejemplo siempre que en la expresión a evaluar introduzcamos variables distintas a *X*, ya que ésta es la que se utiliza en el bucle de las líneas 40 a 60 para buscar los valores de *Y*.

Otros de los problemas a los que nos puede conducir el uso de **VAL**, es a errores del tipo **C Non-**

A partir de la tabla de valores que se calcula en el programa de ejemplo, podemos realizar la representación de la función correspondiente.

x	y
-10	100
-9	81
-8	64
-7	49
-6	36
⋮	⋮
9	81
10	100



sense in BASIC, al intentar evaluar cadenas que no tienen sentido numérico posible; así por ejemplo, un error de este tipo se producirá al ejecutar **PRINT VAL "3*?",** puesto que *3*X* no es una expresión numérica válida. Tengamos en cuenta, que aunque **3** es una constante numérica y el asterisco **(*)** es una operación matemática admitida (multiplicación), el signo de interrogación **(?)** no puede ser considerado ni tan siquiera como parte de una variable.

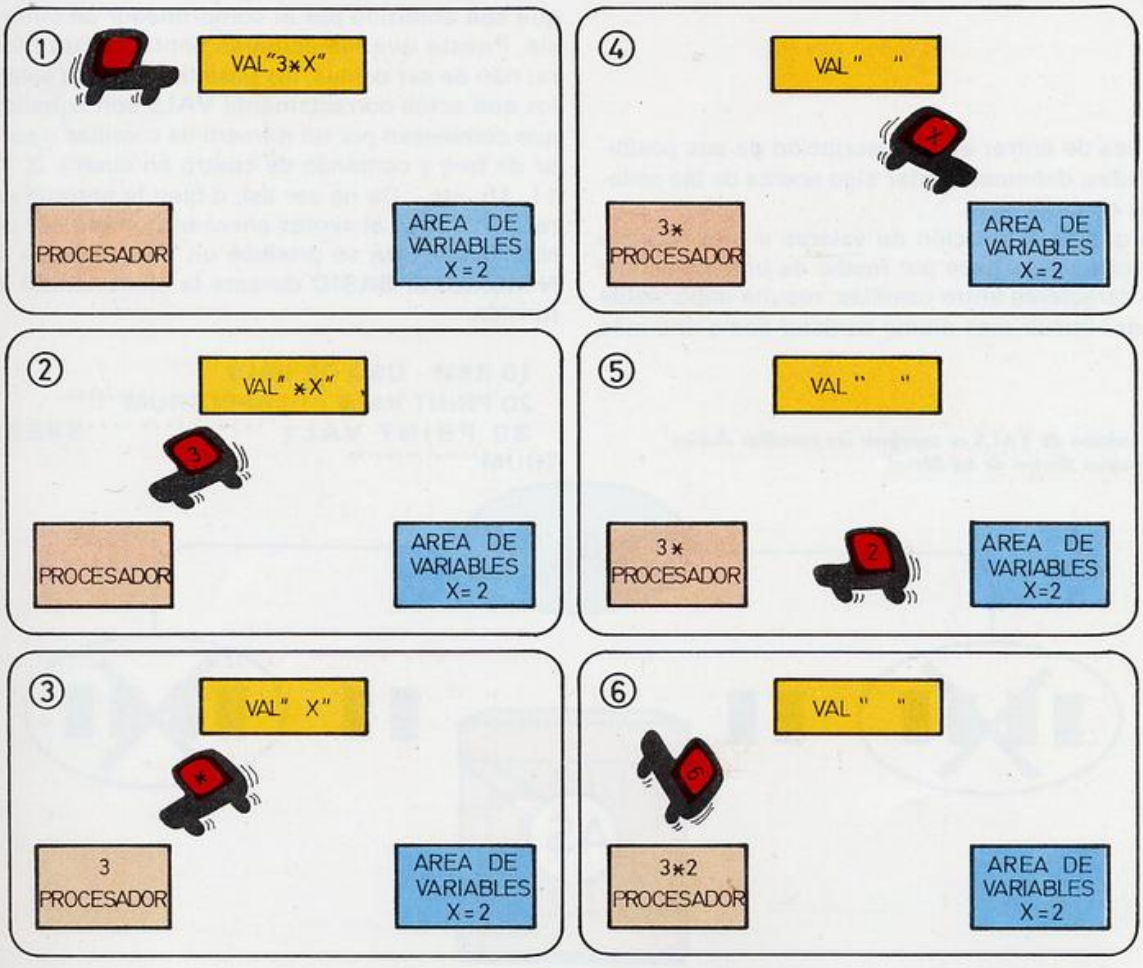
Otros de los problemas que nos pueden surgir con la función que estamos estudiando, son los derivados de la obtención de resultados demasiado grandes o demasiado pequeños, lo que genera errores del tipo **6 Number too big** (número demasiado grande). De entre los problemas de este tipo, probablemente el más frecuente sea el de intentar efectuar divisiones por cero. En nuestro ejemplo, esto puede ocurrir si introducimos la función *Y=X/X,* puesto que en el bucle en el que calcula la tabla de valores, la variable *X* recibe desde –10 hasta 10, con un incremento unitario, pasando por tanto por cero.

El evaluador va algo más lejos que lo expuesto hasta el momento, y es capaz de obtener los valores numéricos de expresiones de cadena que no corresponden exactamente en su formato a una expresión de este tipo. Veamos esto con unos ejemplos; en el formato numérico estándar, los signos aparecen inmediatamente delante del número; del mismo modo, los ceros al comienzo de una cifra nunca se representan.

Una característica de la función **VAL** es la de «normalizar» las expresiones de cadena con el formato estándar de una variable numérica común; para ello, ignora los espacios en blanco que puedan existir entre los elementos de cualquier expresión numérica, así como los ceros a la izquierda de una cifra. Esto nos permite poder efectuar cálculos con variables de cadena, que están preparadas para ser impresas, sin adaptarse concretamente al formato numérico; por ejemplo, con blancos o ceros a la izquierda, sin que ello produzca ningún error. Por otra parte, podemos emplear la función **VAL** para eliminar este tipo de diferencias respecto al formato estándar:

```
10 REM - EJEMPLOS DE VAL
20 CLS
30 LET X$="-   1234":PRINT X$,VAL X$
40 LET X$="-001234":PRINT X$,VAL X$
50 LET X$="0001234":PRINT X$,VAL X$
60 LET X$="- 1234 ":PRINT X$,VAL X$
70 LET X$="    1234":PRINT X$,VAL X$
```

*Diagrama del proceso seguido por el EVALUADOR para el cálculo de la expresión de cadena "3*X".*



BITS

La función **VAL** tiene el cometido inverso al cumplido por **STR$:** calcula el valor numérico de una variable o expresión de cadena.

*

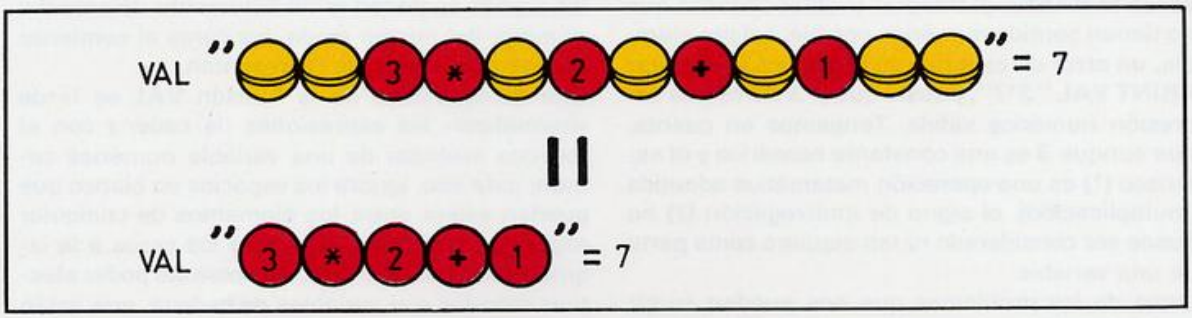
Una característica de la función **VAL,** es la de «normalizar» las expresiones de cadena con el formato numérico estándar.

*

VAL ignora los espacios en blanco que puedan existir entre los elementos de cualquier expresión numérica, así como los ceros a la izquierda de una cifra.

*

La función **VAL$** elimina las dobles comillas de un literal.



La función **VAL** *ignora los espacios en blanco, entre los elementos de una expresión numérica dentro de una cadena.*

¡!

Las funciones pueden admitir como argumento cualquier expresión, siempre que sea del tipo adecuado.

*

La función **STR$** permite obtener la forma de cadena de una expresión numérica.

*

Mediante la función **LEN**, podemos calcular el número de caracteres contenidos en una variable de cadena o expresión del mismo tipo.

En este programa se señalan en la parte izquierda de la pantalla, formas muy diferentes que puede adoptar una cifra, y a partir del centro de la misma, la forma normalizada que se consigue mediante el uso de **VAL**.

LA FUNCION VAL$

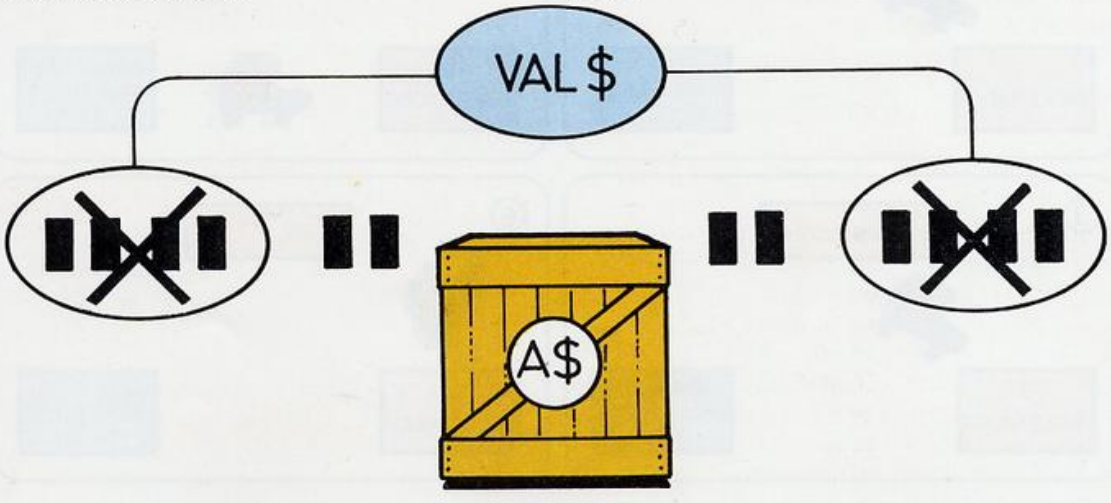
Antes de entrar en la descripción de sus posibilidades, debemos contar algo acerca de las cadenas de caracteres.
Ya que la asignación de valores a una variable de cadena se hace por medio de una secuencia de caracteres entre comillas, resulta impensable poder incluir este último carácter como uno más dentro de la secuencia que compone la *string*. Existe, en cambio, un método que permite incluir este carácter dentro de la secuencia, que consiste en teclear una doble comilla en el lugar deseado, además de las comillas simples habituales de principio y fin del literal. Este es un ejemplo que muestra cómo incluir una palabra «entrecomillada», dentro de las comillas propias de un literal, asignado a una variable:

```
LET X$="ESTA ES UNA PALABRA ""ENTRECOMILLADA"" ENTRE COMILLAS"
```

La función **VAL$**, todo hay que decirlo, es exclusiva del BASIC Sinclair y de muy poco uso. Su misión es eliminar las comillas dobles de una cadena de caracteres, que ha de comenzar y terminar por un número impar de comillas, para que sea admitido por el comprobador de sintaxis. Puesto que las comillas dentro de un literal han de ser dobles, los posibles literales sobre los que actúa correctamente **VAL$** son aquellos que comienzan por un número de comillas a partir de tres y contando de cuatro en cuatro: 3, 7, 11, 15, etc... De no ser así, o bien la entrada es rechazada por el *syntax checker* (número par de comillas) o bien se produce un error del tipo **C Nonsense in BASIC** durante la ejecución de la función:

```
10 REM - USO DE VAL$
20 PRINT VAL$ """"SPECTRUM""""
30 PRINT VAL$ """""""""""""SPECTRUM"""""""""""""
```

La misión de **VAL$** *es suprimir las comillas dobles incluidas dentro de un literal.*



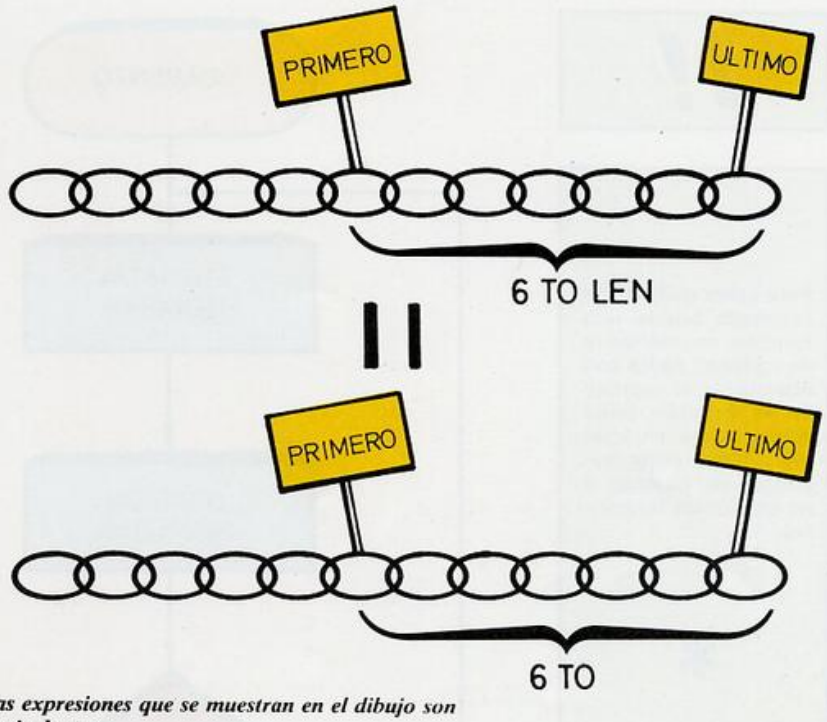
En el ejemplo de la línea 20 obtendremos la palabra sin comilla alguna y en el de la 30 con comillas dobles.
Como podemos observar, cualquier literal debe ir encerrado entre comillas y por tanto, cualquier intento de desproveerle de éstas será rechazado por el ordenador. Así pues, pretender emplear la función **VAL$** sobre un literal sin comillas dobles, abocará a un error del tipo **C Nonsense in BASIC**; tal sería el caso de **PRINT VAL$ "SPECTRUM"**.

UN PEQUEÑO REPASO

Con todos los conocimientos que hemos obtenido hasta el momento, ya estamos capacitados para llevár a cabo un ejercicio de relativa complejidad, basado en el tratamiento de cadenas: realizaremos la toma de 10 nombres y los presentaremos en la pantalla separándolos por cada espacio en blanco que encontremos. Por ejemplo, la entrada ANTONIO RODRIGUEZ MENENDEZ, sería escrita:

ANTONIO
RODRIGUEZ
MENENDEZ

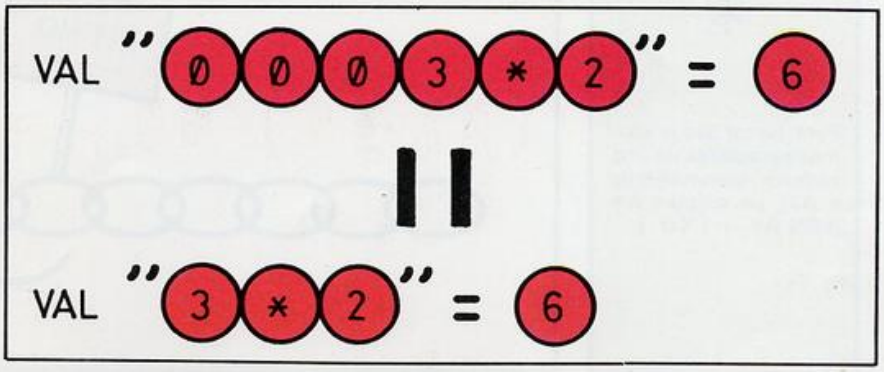
Para separar cada grupo de nombres dejaremos una línea en blanco. Ahora, como es habitual, haremos notar una serie de circunstancias que nos pueden ayudar en la confección del programa. En primer lugar, queda claro que la estructura principal del mismo, es un bucle que realiza la siguiente tarea: toma el nombre y lo separa para presentarlo tal como se requiere y, finalmente, deja el espacio en blanco para diferenciar el próximo nombre.
En cuanto a la separación del nombre en subcadenas, aprovecharemos nuestros conocimientos recién adquiridos sobre la función **LEN** para completar algo de lo dicho en el capítulo anterior sobre el fraccionamiento de *strings;* entonces vimos cómo tomar la parte derecha de una cadena (por ejemplo A$) hasta el final de la misma se efectuaba de la siguiente manera: **A$ (COMIENZO TO)**. Ahora podemos añadir que esta expresión es equivalente a **A$ (COMIENZO TO LEN A$)**. De esto se infiere que una pista acerca de nuestra manera de buscar los espacios en blanco para separar las partes del nombre entrado, puede ser



Las expresiones que se muestran en el dibujo son equivalentes.

utilizar un bucle cuyo punto inicial sea 1 (primer elemento de la cadena) y cuyo final sea la **LEN** de la cadena. Podemos memorizar en una variable el número de carácter en el que se encuentra el último blanco detectado, para así tomar la siguiente subcadena desde este punto hasta el próximo blanco.
Aunque no esté directamente relacionado con el programa que proponemos, aprovecharemos la ocasión para destacar que en ciertos casos es útil conocer los *n* últimos caracteres de una cadena. A tal fin emplearemos la función **LEN**, operando bajo la siguiente fórmula general: *n* últimos caracteres = A$ (LEN A$–*n*–1 TO). Así, por ejemplo, si deseamos conocer los cuatro últimos caracteres de la cadena A$, deberíamos escribir **PRINT**

La normalización del formato numérico que **VAL** *lleva a cabo, incluye la supresión de los ceros innecesarios.*



¡!

Para saber qué tipo de resultado brinda una función (numérico o de cadena), basta con advertir si el nombre de la función tiene como último carácter un símbolo dólar (resultado de cadena), o no (resultado numérico).

*

Los posibles literales sobre los que actúa correctamente **VAL$**, son aquellos que comienzan y terminan por un número de comillas a partir de tres y contando de cuatro en cuatro (3, 7, 11, 15...).

*

La expresión VARIABLE$ (COMIENZO TO) es equivalente a VARIABLE$ (COMIENZO TO LEN VARIABLE$).

*

Para tomar los *n* últimos caracteres de una cadena (generalizada a A$), se emplea **A$ (LEN A$-*n*-1 TO)**.

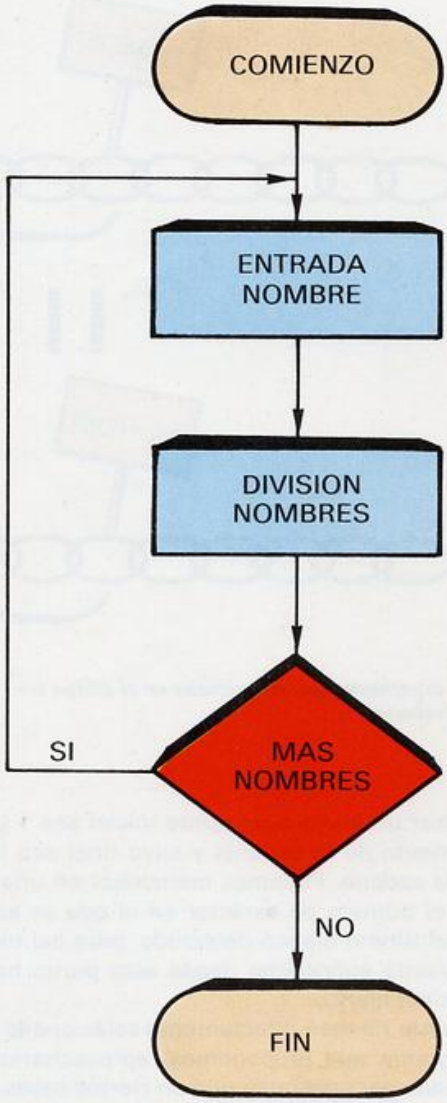


Diagrama de flujo del programa "DIVISION DE NOMBRE".

La función* LEN *se puede emplear para realizar un* slicing *inverso, es decir, de derecha a izquierda de una cadena.

A$ (LEN A$-3 TO). Por supuesto, se generará un error en caso de que la cadena A$ contenga menos de cuatro caracteres.

Bien, ha llegado el momento de que intentemos hacer nuestro propio programa:

```
10 REM DIVISION DE NOMBRES
20 CLS
30 FOR I=0 TO 9
40 INPUT "NOMBRE?";N$: LET P=1
50 FOR J=1 TO LEN N$
60 IF N$(J)=" " THEN PRINT N$(P TO J): LET P=J+1
70 NEXT J
80 PRINT
90 NEXT I
```

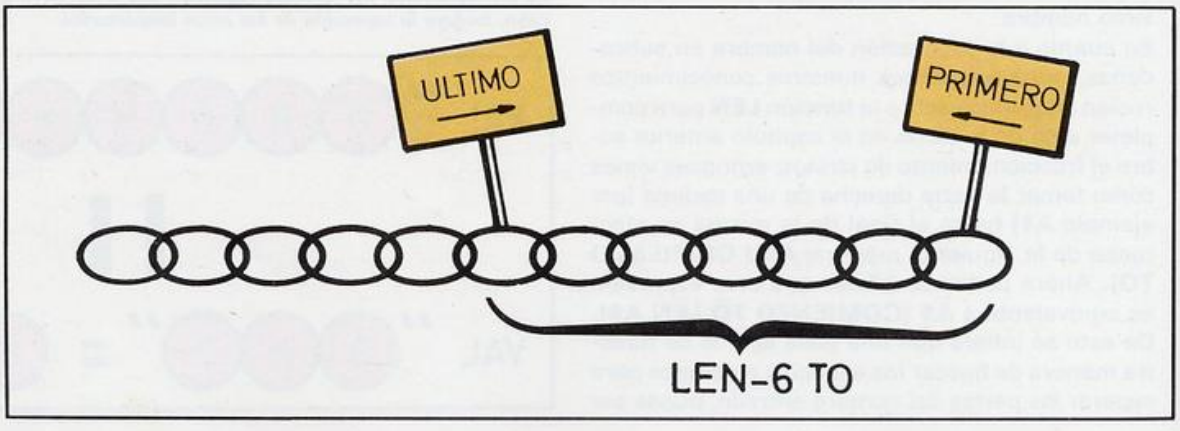
Las líneas 10 y 20 forman la zona de inicializaciones, donde se da título al programa y se borra la pantalla para la presentación más clara de los resultados.

En las líneas 30 y 90 se configura la estructura de bucle basada en la variable I, que consigue que el bloque comprendido entre la línea 40 y la 80 sea repetido 10 veces (de 0 a 9).

En la zona de programa que comprende de la línea 40 a la 80, se realiza el tratamiento de cada nombre. Este comienza por la entrada del mismo a la variable N$ mediante un **INPUT** (línea 40) y finaliza con la impresión de la línea de separación para preparar el tratamiento de un nuevo nombre (línea 80).

Por tanto, la parte más importante de nuestro programa, la cual realiza la división del nombre y la impresión de sus diversas partes, se encuentra en el bucle anidado desde las líneas 50 a la 70. Este bucle de segundo nivel, controlado por la variable J, recorre la cadena entrada (N$) buscando un espacio en blanco (línea 60). Si no lo localiza en el carácter estudiado, continúa analizando el siguiente carácter, y así sucesivamente hasta el final de la cadena.

Si por el contrario localiza un espacio en blanco, escribe la parte de cadena analizada hasta el momento y hace que la variable P indique el último punto tomado como blanco.



SISTEMAS DE NUMERACION

CUALQUIER proceso informático, es decir, de tratamiento de la información, consta básicamente de tres etapas: entrada de los datos, proceso de los mismos y representación de los resultados obtenidos.

Los datos pueden ir desde algo tan simple como dos cantidades a sumar, hasta algo tan complicado como la información obtenida a partir de sensores que midan la humedad del aire, la temperatura, la presión atmosférica, etc., con el objetivo de establecer un pronóstico del tiempo. El proceso consistirá en aquella serie de operaciones que han de ejecutarse, para poder obtener un resultado adecuado a nuestras necesidades.

Al igual que los datos, los resultados podrán ser algo sencillo, como la cantidades suma de las dos anteriores, o algo complicado, como un gráfico de líneas de alta y baja presión, con anticiclones y borrascas, para pronosticar el tiempo de mañana o las próximas semanas.

Una mirada a nuestro Spectrum, puede sugerirnos la pregunta de cómo un aparato tan pequeño es capaz de ejecutar infinidad de labores, tan distintas unas de otras. Todo lo anterior, conduce a la idea de ordenador como máquina inteligente, capaz de «pensar» por sí misma. Es más, hasta hace no mucho tiempo, los ordenadores eran denominados cerebros electrónicos.

En realidad, esto no es así. Ni el Spectrum piensa como nosotros, ni es tan inteligente como parece. No es capaz de comprender nuestro lenguaje, y las órdenes que le mandemos ejecutar, antes de almacenarlas en su memoria debe traducirlas a un alfabeto propio, formado únicamente por números.

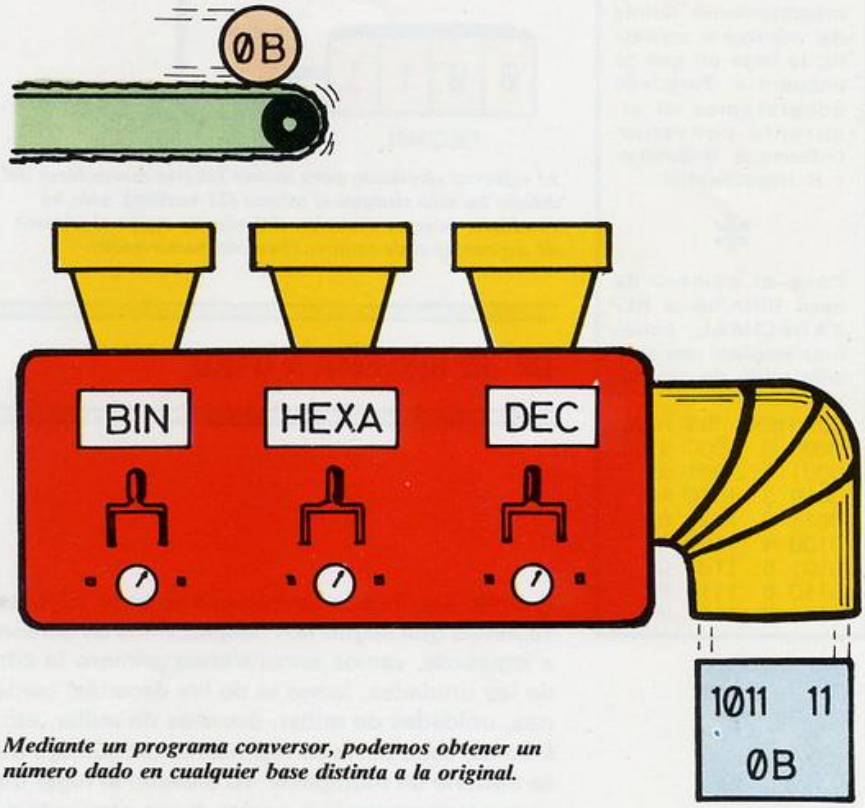
Ahora bien, nosotros estamos acostumbrados a manejar cantidades numéricas en base 10; es decir, para representar un determinado guarismo, formamos una combinación con las cifras que van del 0 al 9. Ni siquiera eso es capaz de reconocer nuestro micro.

A nadie se le oculta que la electricidad es lo que da vida a nuestro ordenador. Y es aquí donde reside el secreto. Cuando el Spectrum trata de acceder a cualquier información almacenada en la memoria, su particular forma de pensar, sólo sirve para reconocer una cosa: si allí hay corriente o no.

Un convenio adoptado en informática, diferencia entre sí estas dos posibilidades, asignándoles unos valores. Cuando hay corriente, decimos que allí hay un 1. Cuando no la hay, diremos que está a 0. Esto es lo que se denomina lógica binaria, o sistema binario para la representación de la información. Es decir, entre los distintos componentes por los que circula la información dentro de nuestro micro, sólo podemos encontrar dos estados posibles (1 ó 0). Esta es la unidad mínima de información y se denomina BIT *(BInary digiT,* dígito binario).

Con un solo bit, no podemos conseguir mucho. Por ello, se asocian en grupos de ocho para formar unidades superiores llamadas BYTES. Tenemos, por tanto, 2↑8=256, combinaciones diferentes de unos y ceros, que representarán distintos caracteres (letras, números, caracteres especiales o de control, etc...). Estas 256 posibilidades, configuran el alfabeto privado del Spectrum. Se trata del denominado código A.S.C.I.I.

Este código es el más utilizado en la comunicación entre periféricos, así como para la representación de caracteres y datos en la memoria central de la mayoría de microordenadores.



Mediante un programa conversor, podemos obtener un número dado en cualquier base distinta a la original.

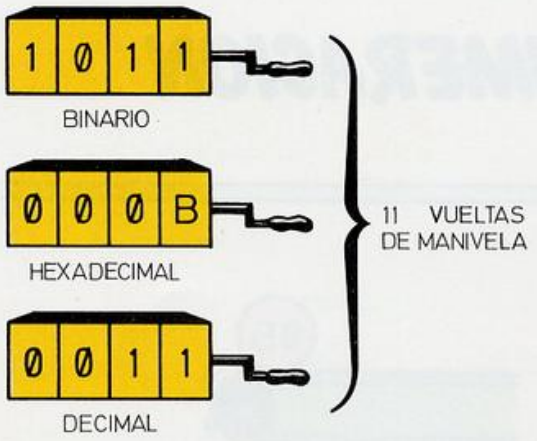
¡!

Para utilizar el programa de conversión, deberemos indicar, inmediatamente detrás del número a convertir, la base en que se encuentra. Para ello adoptaremos el siguiente convenio: D=Decimal, B=Binario, y H=Hexadecimal.

*

Para el cambio de base BINARIA a HEXADECIMAL, podemos emplear una sencilla tabla de conversión:

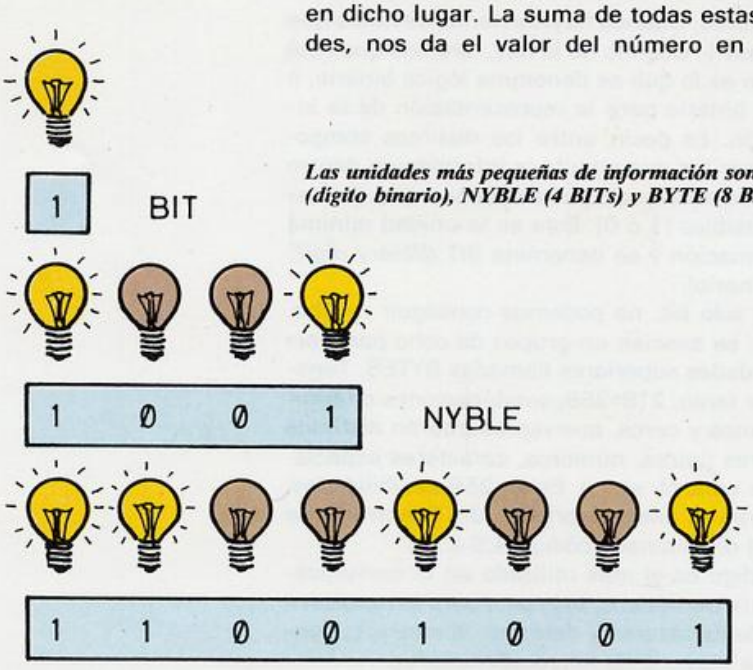
BIN.	HEX.	BIN.	HEX.
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F



El esfuerzo efectuado para mover los tres marcadores del dibujo ha sido siempre el mismo (11 vueltas), sólo ha cambiado la representación del mismo, según el número de dígitos de cada tambor (base de numeración).

DE UN SISTEMA A OTRO

Cuando escribimos un número en base 10, todos sabemos que según nos desplazamos de derecha a izquierda, vamos encontrando primero la cifra de las unidades, luego la de las decenas, centenas, unidades de millar, decenas de millar, etc... Esto es, cada posición representa un número que se obtiene de multiplicar 10 elevado al lugar que ocupa (comenzando a contar desde cero y de derecha a izquierda), por la cifra que se encuentra en dicho lugar. La suma de todas estas cantidades, nos da el valor del número en cuestión.



Las unidades más pequeñas de información son: BIT (dígito binario), NYBLE (4 BITs) y BYTE (8 BITs).

Con un ejemplo concreto lo veremos más claramente. Supongamos que hemos escrito 32956. La cifra 6, ocupa el primer lugar comenzando por la derecha. Tendremos entonces 6*10↑0=6*1=6. El siguiente dígito es un 5. Por tanto, 5*10↑1=5*10=50. El mismo proceso se sigue con los demás: 9*10↑2=9*100=900. 2*10↑3=2*1000=2000 y, finalmente, 3*10↑4=3*10000=30000. Ahora sumamos las cantidades que hemos ido obteniendo: 6+50+900+2000+30000 y el resultado es el número que habíamos tomado en el ejemplo, 32956. Fácil, ¿verdad?

Cualquier otro sistema de numeración, sigue el mismo procedimiento. Lo único que cambiará será la base.

Como vimos antes, el sistema binario o en base dos, tan solo dispone de dos dígitos, el 0 y el 1. Es decir, un número expresado en esta base, estará formado por una combinación de ceros y unos.

Sea, por ejemplo, el número binario 00101011. Recordemos que la base ahora es 2. Por tanto, comenzando por la derecha tenemos: 2↑0*1=1*1=1, 2↑1*1=2*1=2, 2↑2*0=4*0=0, 2↑3*1=8*1=8, 2↑4*0=16*0=0, 2↑5*1=32*1=32, 2↑6*0=64*0=0 y, para terminar, 2↑7*0=128*0=0. Sumando todos los resultados, 1+2+8+32, hallamos que 43 es el correspondiente valor decimal del número binario 00101011. Como vemos, añadir ceros por la izquierda, no altera el valor del número en cuestión.

La conversión binario-decimal, se efectúa en el Spectrum mediante la función BIN. Si ejecutamos la instrucción **PRINT BIN 00101011** la respuesta será sin duda, 43.

Para realizar el paso contrario, decimal a binario, se utiliza el método de las divisiones sucesivas, entre 2. Es decir, tomamos el número decimal y lo dividimos por dos. Anotamos el resto obtenido. Si el cociente es igual o mayor de dos, volvemos a dividir. Señalamos, de nuevo, el resto. El proceso continúa hasta que resulte el cociente menor que dos. El equivalente binario se construye tomando estos números de abajo a arriba, comenzando por el último cociente, y anotándolos de izquierda a derecha. El cuadro adjunto muestra gráficamente el proceso seguido.

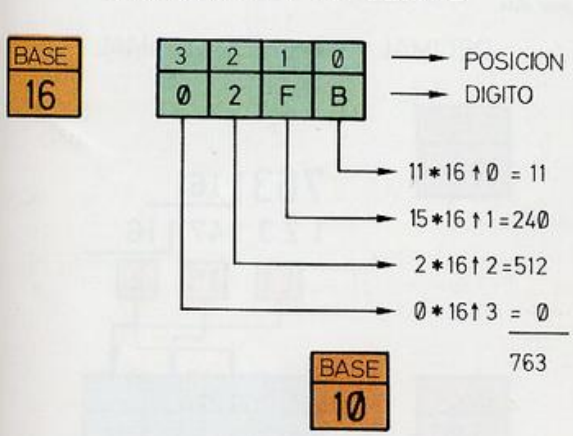
UNA NUEVA BASE

Quizás alguna vez, hayamos asistido atónitos a listados de programas escritos en código máqui-

na, que utilizan una extraña notación, mezcla de números y letras. Y bien, ¿no habíamos quedado en que nuestro ordenador sólo era capaz de identificar dígitos binarios (bits)? Entonces, si el código máquina es lo más aproximado al lenguaje de nuestro Spectrum ¿por qué no está formado por unos y ceros?

La respuesta es sencilla. Imaginemos un listado de miles de instrucciones, compuesto únicamente por ceros y unos. ¡A ver quién es el valiente que interpreta eso!

Para facilitar la labor de introducción del código máquina, acortando la longitud de sus cifras, se emplea otro sistema de numeración, en el que probablemente hemos visto los listados antes citados: el hexadecimal, que expresa los números en base 16. Mediante este sistema de numeración podemos expresar números más grandes con menor cantidad de dígitos. Además, como

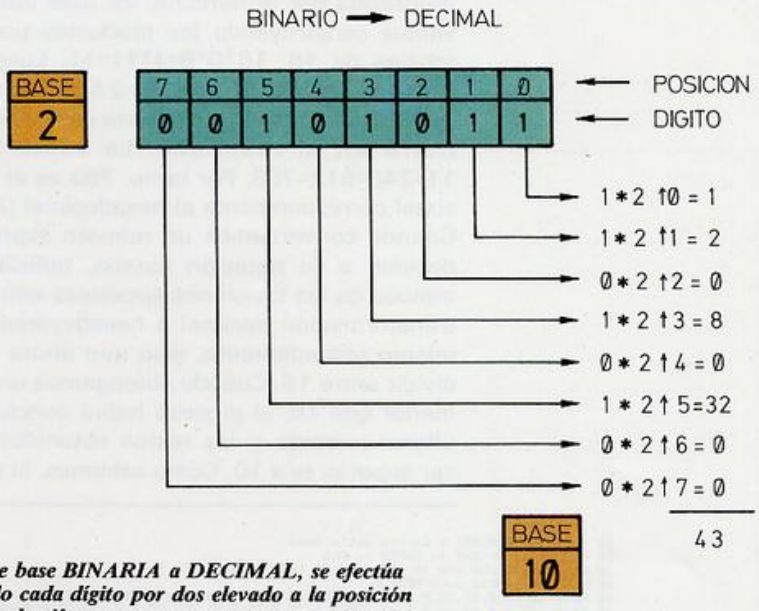


El cambio de base HEXADECIMAL a DECIMAL, se efectúa multiplicando cada dígito por dieciséis elevado a la posición que ocupa en la cifra.

tendremos oportunidad de comprobar más adelante, es muy fácil la conversión del sistema hexadecimal al binario.

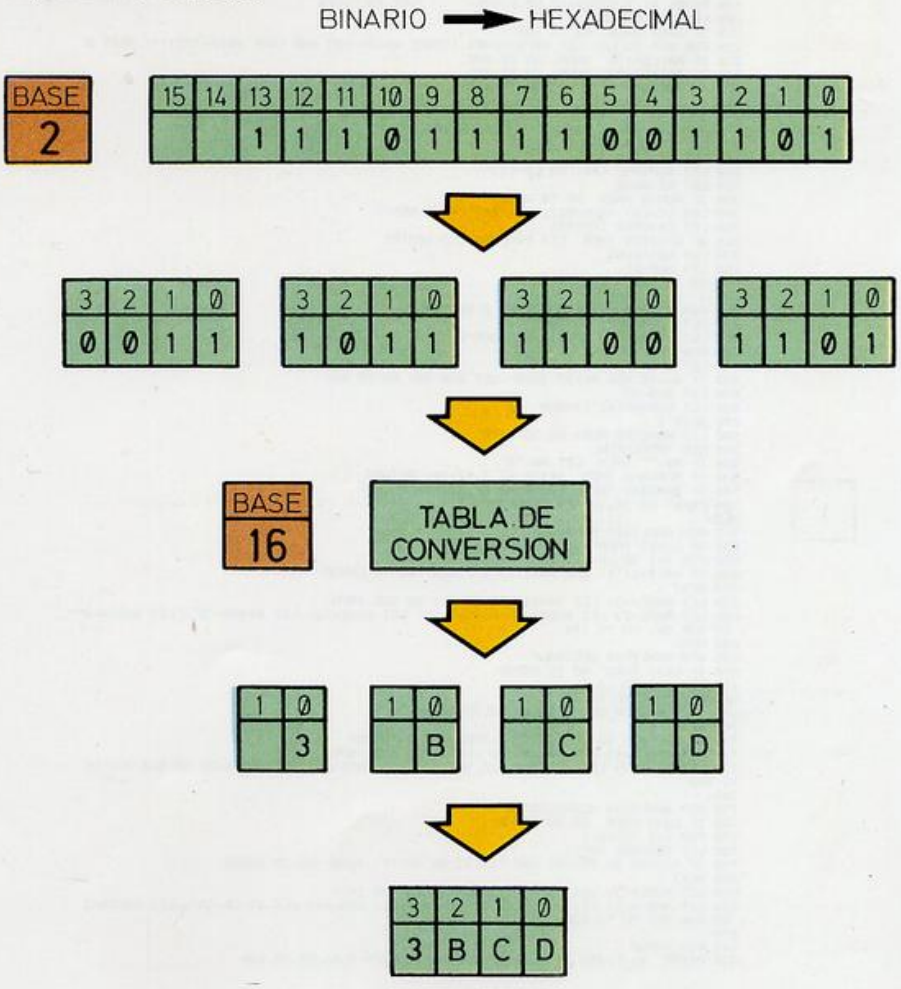
El sistema binario dispone de dos dígitos, 0 y 1. El decimal de 10, del 0 al 9. Por tanto, en hexadecimal, tendremos 16 dígitos, que toman valores de 0 a 15. Y aquí es donde surge el problema. Estamos hablando de dígitos, es decir, una sola cifra para cada posición dentro del número. De 0 a 9, no hay ninguna ambigüedad, pero para representar más de 10 necesitamos dos cifras. La cuestión se solventa asginando a los valores que van del 10 al 15 las primeras letras del alfabeto. Según esto, en vez de 10, escribimos A. El 11, es una B, el 12 una C, el 13 una D, el 14 una E y el 15 una F.

La forma de tratar los números en base 16, es idéntica a como lo hacíamos con los sistemas binario y decimal. Sea, por ejemplo, el número 02FB, en base 16. Tomamos la primera cifra co-



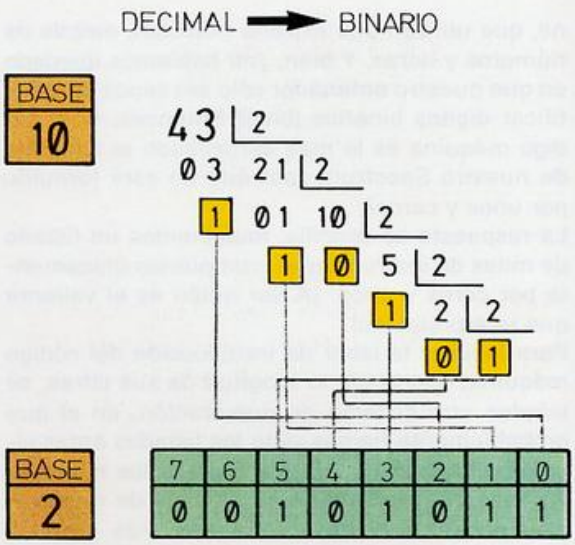
El cambio de base BINARIA a DECIMAL, se efectúa multiplicando cada dígito por dos elevado a la posición que ocupa en la cifra.

Para el cambio de números en base BINARIA a HEXADECIMAL, resulta útil dividir la cifra completa en grupos de 4 BITs (NYBLE), para así poder utilizar una simple tabla de conversión.

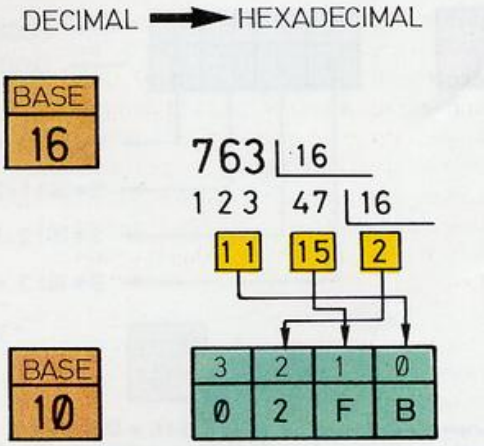


menzando por la derecha, en este caso la B, y vamos construyendo los productos por las potencias de 16: 16↑0*B=1*11=11. Luego la segunda: 16↑1*F=16*15=240. La tercera: 16↑2*2=256*2=512. La cuarta es 0, luego no influirá en el resultado. La suma total es: 11+240+512=763. Por tanto, 763 es el valor decimal correspondiente al hexadecimal 02FB.

Cuando convertíamos un número expresado en decimal a su notación binaria, utilizábamos el método de las divisiones sucesivas entre dos. La transformación decimal a hexadecimal sigue el mismo procedimiento, sólo que ahora debemos dividir entre 16. Cuando obtengamos un cociente menor que 16, el proceso habrá concluido. Este último cociente o los restos obtenidos, pueden ser superiores a 10. Como sabemos, si este es el



Para la conversión de un número en base DECIMAL a BINARIA, se emplea el sistema de divisiones sucesivas por dos.



Para la conversión de un número en base DECIMAL a HEXADECIMAL, se emplea el sistema de divisiones sucesivas por dieciséis.

```
 10 REM HEXA.BIN.DEC $ Carlos de la Ossa
 20 BORDER 4: BRIGHT 1: PAPER 4: CLS
 30 PLOT 35,0: FOR n=0 TO 23: IF n=12 THEN  PLOT 40,5
 40 READ x,y: DRAW x,y: NEXT n
 50 DATA 105,0,0,35,35,0,0,105,-35,0,0,35,-105,0,0,-35,-35,0,0,-105,35,0,0,-35
 60 DATA 175,0,0,35,35,0,0,95,-35,0,0,35,-175,0,0,-35,-35,0,0,-95,35,0,0,-35
 70 PLOT 17,17: FOR n=0 TO 7: IF N=4 THEN  PLOT 19,19
 80 READ x,y: DRAW x,y: NEXT n
 90 DATA 221,0,0,141,-221,0,0,-141
100 DATA 217,0,0,137,-217,0,0,-137
110 PLOT 25,45: DRAW 205,0: DRAW 0,85: DRAW -205,0: DRAW 0,-85
120 PLOT 26,46: DRAW 203,0: DRAW 0,83: DRAW -203,0: DRAW 0,-83
130 PRINT AT 3,11; BRIGHT 1;"CONVERSOR"
140 PRINT AT 7,4;"DECIMAL:": PRINT AT 10,4;"BINARIO:": PRINT AT 13,4;"HEXADECIM
AL:"
150 LET R$="                "
160 POKE 23658,8: POKE 23609,20: POKE 23562,1
170 LET DB=288: LET DH=288: LET BD=410: LET HD=410: LET IMPR=510: LET ANB=570:
LET AND=650: LET ANH=750: LET ERROR=840
180 INPUT ;"NUMERO ? "; LINE A$
190 PRINT AT 7,12;R$;AT 10,12;R$+"    ";AT 13,16;R$
200 LET LG=LEN A$
210 IF LG=0 THEN  GO TO 180
220 FOR N=1 TO LG: LET A$(N)=CHR$ ((CODE A$(N)-(32 AND CODE A$(N))>72))): NEXT N
230 IF A$(LG)="B" THEN  GO TO ANB
240 IF A$(LG)="D" THEN  GO TO AND
250 IF A$(LG)="H" THEN  GO TO ANH
260 GO TO ERROR
270 :
280 REM DECIMAL A BINARIO Y HEXADECIMAL
290 LET LG=LEN A$
300 LET N$=""
310 LET NUM=VAL (A$( TO LG-1))
320 LET BS=BASE
330 IF NUM=0 THEN  GO TO IMPR
340 LET CC=INT (NUM/BS): LET RESTO=NUM-BS*CC
350 LET C$=STR$ (RESTO)
360 IF RESTO>9 THEN  LET C$=CHR$ (55+RESTO)
370 LET N$=C$+N$
380 LET NUM=CC
390 GO TO 330
400 :
410 REM BINARIO Y HEXADECIMAL A DECIMAL
420 LET BS=BASE
430 LET D$=A$( TO LG-1): LET NUM=0
440 FOR I=0 TO LG-2
450 LET A=CODE (D$(LG-1-I))
460 IF A>=48 AND A<=57 THEN  LET A=A-48: GO TO 480
470 LET A=A-55
480 LET NUM=A*(BS^I)+NUM
490 NEXT I
500 LET N$=STR$ NUM: GO TO IMPR
510 REM IMPRESION
520 IF N$="" THEN  LET N$="0"
530 IF MARCA=1 THEN  PRINT AT 7,12;N$; RETURN
540 IF MARCA=2 THEN  PRINT AT 10,12;N$; RETURN
550 PRINT AT 13,16;N$; RETURN
560:
570 REM ANALISIS BINARIO
580 IF LG>17 THEN  GO TO ERROR
590 FOR I=1 TO LG-1
600 IF A$(I)<>"1" AND A$(I)<>"0" THEN  GO TO ERROR
610 NEXT I
620 LET MARCA=2: LET N$=A$( TO LG-1): GO SUB IMPR
630 LET BASE=2: LET MARCA=1: GO SUB BD: LET BASE=16: LET A$=N$+"D": LET MARCA=3
: GO SUB DH: GO TO 180
640 REM
650 REM ANALISIS DECIMAL
660 IF LG>6 THEN  GO TO ERROR
670 FOR I=1 TO LG-1
680 LET CG=CODE A$(I)
690 IF CG<48 OR CG>57 THEN  GO TO ERROR
700 NEXT I
710 IF VAL A$( TO LG-1)>65535 THEN  GO TO ERROR
720 LET MARCA=1: LET N$=A$( TO LG-1): GO SUB IMPR
730 LET BASE=2: LET MARCA=2: GO SUB DB: LET BASE=16: LET MARCA=3: GO SUB DH: GO
TO 180
740:
750 REM ANALISIS HEXADECIMAL
760 IF LG>5 THEN  GO TO ERROR
770 FOR I=1 TO LG-1
780 LET CG=CODE A$(I)
790 IF (CG<48 OR CG>57) AND (CG<65 OR CG>70) THEN  GO TO ERROR
800 NEXT I
810 LET MARCA=3: LET N$=A$( TO LG-1): GO SUB IMPR
820 LET MARCA=1: LET BASE=16: GO SUB HD: LET BASE=2: LET A$=N$+"D": LET MARCA=2
: GO SUB DB: GO TO 180
830:
840 REM ERROR
850 PRINT  0; FLASH 1;"ENTRADA ERRONEA": BEEP 3,0: GO TO 180
```

caso, sustituiremos dichos valores por las letras correspondientes. En el cuadro se ilustra el proceso seguido.

El paso de binario a hexadecimal y viceversa, se efectúa en dos etapas. En la primera convertimos nuestro número a su equivalente decimal, y desde aquí a la base que deseemos, siguiendo los métodos descritos anteriormente. Es conveniente, cuando se trate de números binarios con gran cantidad de cifras, separarlas en grupos de cuatro, comenzando por la derecha. Si el último grupo tiene menos de cuatro dígitos, añadimos el número de ceros necesarios, a la izquierda, hasta completarlo. El ejemplo del cuadro aclarará las ideas.

El proceso de conversión entre los distintos sistemas, realizado a mano, puede resultar largo y tedioso. Nuestro Spectrum, será una gran ayuda. El siguiente programa realiza todo el trabajo.

LA BOMBA

POR una de esas «casualidades» de la vida, nos encontramos cumpliendo nuestro Servicio Militar. Estamos encuadrados en la SERECO (Sección de Reconocimiento), perteneciente a la compañía de armas de apoyo del II Batallón del Regimiento de Infantería más aguerrido de toda la Nación.
Aparentemente es un día como otro cualquiera: luce el sol, nos refresca una suave brisa, los hombres del I Batallón piden a voz en grito «más instrucción...»: una jornada muy normal.
De repente, sin apenas darnos cuenta, las sirenas de alarma comienzan a ulular; un intenso escalofrío y una breve sensación de ignorancia nos recorre el cuerpo. ¿Qué pasa? Casi al instante empezamos a correr hacia la compañía con todas las fuerzas y velocidad que nos daban nuestras piernas. Los rumores eran ciertos. ¡¡Los visitantes extraterrestres han invadido la Tierra y rodeado nuestro cuartel, situando cargas explosivas en la fachada principal del Regimiento!!
¡Esto era misión para la SERECO! El Teniente Korsy y el Sargento Multus, mandos de dicha sección, cubiertos por el fuego cruzado de las ametralladoras y subfusiles que con habilidad supina manejaban sus hombres, desactivaron con éxito cada uno de los artefactos que los lagartos extraterrestres habían colocado.
Pero... ¿Cómo lo lograron?, ¿qué método utilizaron para desactivar las claves? Eso es precisamente lo que debemos intentar averiguar con este programa.

COMO JUGAR

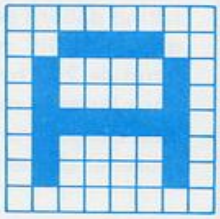
El Spectrum «piensa» una palabra y nosotros tenemos que adivinarla, introduciendo en cada secuencia del juego una letra. Debemos tener cuidado, porque por cada intento errado, el brazo extensible del detonador se acercará cada vez más a la mecha de la bomba. En el momento que se falle por octava vez, el artefacto quedará irreversiblemente activado y listo para explosionar.

Si no hemos conseguido encontrar la clave, nuestro Spectrum hará gala de su caballerosidad mostrándonos la palabra de nuestras desdichas.
Como un detalle más del programa, para facilitar nuestra tarea, cada vez que realizamos un intento, la letra utilizada queda impresa en la parte inferior de la pantalla, recordándonos que no debe volver a utilizarse.

EL PROGRAMA

El programa utiliza la totalidad de los gráficos de usuario, de los cuales veinte definen la calavera

La subrutina generadora de caracteres gigantes, amplía el patrón habitual de un carácter del Spectrum, tanto en anchura como en altura. En el gráfico se muestra una ampliación de 3×2.



¡!

Los caracteres que aparecen con subrayado simple en el listado, corresponden a los gráficos de las teclas cuyo carácter se subraya.

*

Los gráficos editados junto con **CAPS SHIFT**, se representan mediante la tecla correspondiente doblemente subrayada.

¡!

Para la necesaria carga del código máquina, es imprescindible que el programa sea ejecutado desde la línea 1650.

*

El código máquina de la subrutina de caracteres gigantes, ocupa 277 bytes a partir de la dirección 32256.

*

El diseño del programa asegura que las palabras propuestas por el Spectrum sean escogidas al azar y sin repetición, hasta que el programa no es interrumpido o cargado nuevamente.

*

El código máquina debe estar situado en la cinta inmediatamente después del listado BASIC.

que aparece inscrita en la bomba, y el restante, el brazo extensible que porta el detonador. Por tanto, conviene recordar que los caracteres que aparecen subrayados en el listado, equivalen a los gráficos de las teclas correspondientes; por otra parte, siguiendo nuestra norma en materia de listados, los caracteres doblemente subrayados (sólo podrán ser números), corresponden a los gráficos cambiados de dichas teclas, es decir, **CAPS SHIFT** + tecla (en modo gráfico).

En cuanto al fichero interno que contiene las 50 palabras que puede elegir al azar el Spectrum, puede ser ampliado siempre que en la matriz **D** definida en la línea 80, realicemos la pertinente actualización; del mismo modo, también será necesario ampliar el rango de elección del **RND** (línea 650).

Tal y como ha sido diseñado el programa, las palabras se seleccionan aleatoriamente y sin repetición. Debido a esto, puede darse el caso de que al agotarse el banco de palabras, el programa se detenga con el mensaje **E Out of DATA.** En este caso deberemos efectuar **RUN,** para reciclar todo el vocabulario.

A la hora de grabar el programa, lo haremos mediante el uso combinado de los comandos SAVE y LINE de la siguiente forma:

```
SAVE "BOMBA" LINE 1650
```

Para una mayor estética del programa, hemos utilizado la subrutina de caracteres gigantes de PSION Computers, incluida en la casete de demostración de los primeros Spectrum. Dicha subrutina deberá ser grabada a continuación del pograma BASIC con la dirección 32256 y una longitud de 277 bytes.

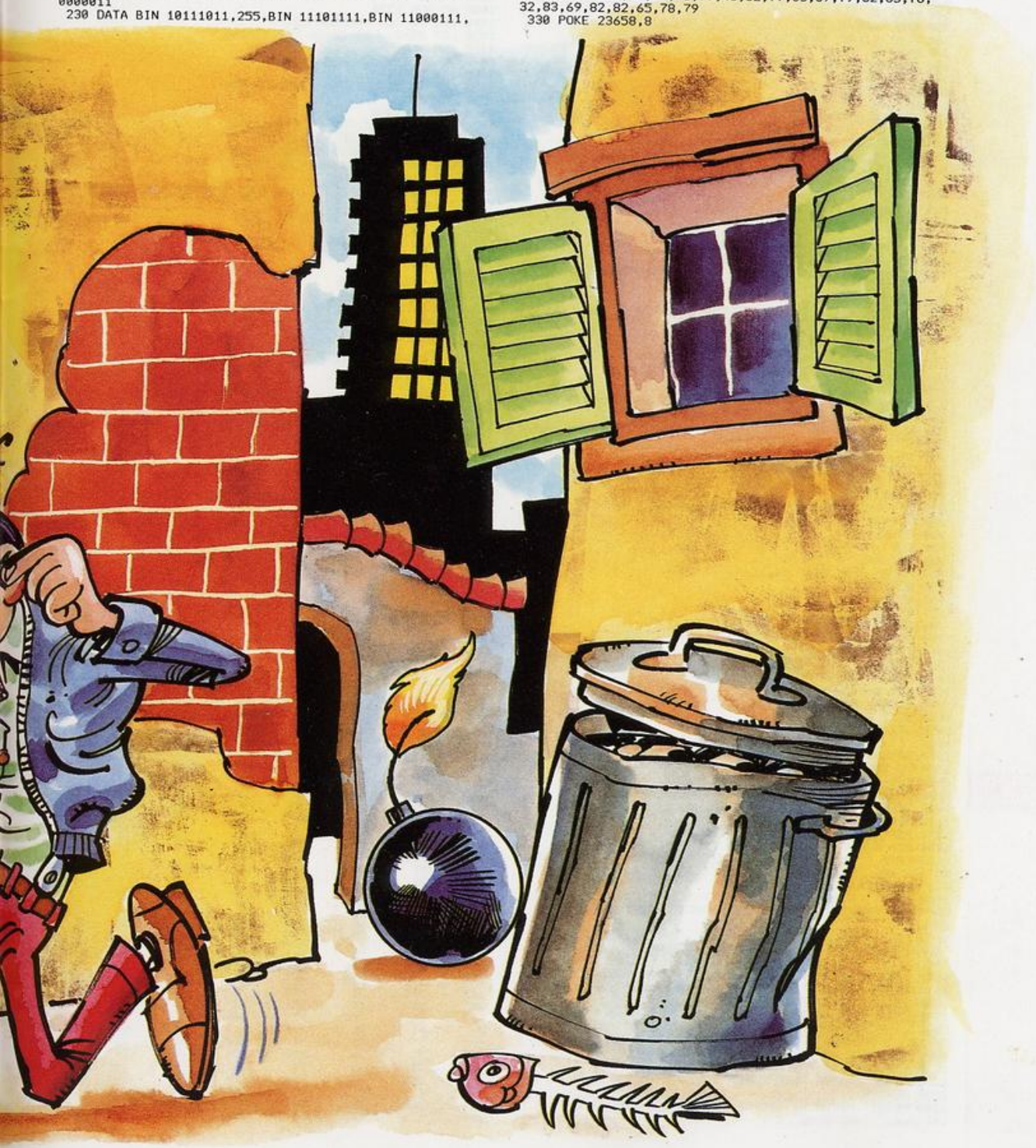
Para llevar a cabo esta última parte de la adopción del programa principal, introduzca en la memoria el siguiente programa:

```
10 REM CARGADOR DE CODIGO MAQUINA
20 CLEAR 32255
30 FOR I=32256 TO 32532
40 INPUT A
50 POKE I,A
60 NEXT I
70 PRINT "SITUA LA CINTA AL FINAL DEL BASIC"
80 SAVE "LIT" CODE 32256,277
```

Este programa realizará la petición de los 277 números correspondientes al listado de código máquina (introduzcámoslos por filas); una vez que haya finalizado la entrada de datos, se preparará la grabación del código máquina, para lo cual deberemos situar la cinta inmediatamente a continuación del final del BASIC anteriormente grabado (listado BASIC del programa LA BOMBA).

```
 10 REM ************************
 20 REM * J.M.MAYORAL SERRANO *
 30 REM ************************
 40 LET SW=0
 50 REM GRAFICOS DEF.
 60 LET ABDUL=257: PAPER 0: BORDER 0: CLS
 70 LET G$="ABCDFGHJKLMNOPQRSTUEI"
 80 DIM D(50)
 90 PRINT  FLASH 1; PAPER 5; INK 1;AT 10,3;"   UN MO
MENTO POR FAVOR   "
 100 PRINT  INK 5, PAPER 1; FLASH 1;AT 12,6;" ~GRAFIC
OS USUARIO ~"
 110 DATA BIN 00000111,BIN 00001111,BIN 00001111,BIN
00000111,BIN 01110110,BIN 11111111,BIN 11111111,BIN 0
1111011
 120 DATA BIN 0,BIN 10000000,BIN 10000000,BIN 0,BIN 0
,BIN 10000000,BIN 11000000,BIN 11100000
 130 DATA BIN 0,BIN 1,BIN 1,BIN 0,BIN 0,BIN 1,BIN 11,
BIN 111
 140 DATA BIN 11100000,BIN 11110000,BIN 11110000,BIN
11100000,BIN 01101110,BIN 11111111,BIN 11111111,BIN 1
```

```
1011110
 150 DATA BIN 11110000,BIN 11111000,BIN 01111100,BIN
00111110,BIN 00011101,BIN 00001011,BIN 00000011,BIN 1
11
 160 DATA 0,0,BIN 00111100,BIN 01111110,255,255,255,2
55
 170 DATA BIN 1111,BIN 11111,BIN 00111110,BIN 0111110
0,BIN 10111000,BIN 11010000,BIN 11000000,BIN 11100000
 180 DATA BIN 111,BIN 1111,BIN 1111,BIN 00011110,BIN
00011100,BIN 00001100,BIN 00000110,BIN 00001011
 190 DATA 255,255,255,254,BIN 011111100,BIN 00111000,
BIN 00010000,BIN 00010001
 200 DATA BIN 11100000,BIN 11110000,BIN 11110000,BIN
11111000,BIN 01111000,BIN 01110000,BIN 11100000,BIN 1
1010000
 210 DATA 0,0,0,0,BIN 00011101,BIN 001111111,BIN 0011
1111,BIN 00111111
 220 DATA BIN 00011011,BIN 00111101,BIN 01111101,BIN
11111011,BIN 11110011,BIN 11100111,BIN 11000111,BIN 1
0000011
 230 DATA BIN 10111011,255,BIN 11101111,BIN 11000111,
BIN 10000011,255,255,255
 240 DATA BIN 11011000,BIN 10111100,BIN 10111110,BIN
11011111,BIN 11001111,BIN 11100111,BIN 11100011,BIN 1
1000001
 250 DATA 0,0,0,0,BIN 10110000,BIN 1111100,BIN 111111
00,BIN 10111100
 260 DATA BIN 00011101,1,7,BIN 1111,BIN 1111,BIN 0000
0110,0,0
 270 DATA BIN 10000001,BIN 11000001,BIN 11000001,BIN
11000000,BIN 10000000,0,0,0
 280 DATA BIN 10011001,BIN 10011001,BIN 10011001,0,25
5,BIN 01111110,BIN 00111100,0
 290 DATA BIN 10000001,BIN 10000011,BIN 10000011,BIN
00000011,BIN 10000001,0,0,0
 300 DATA BIN 10111100,BIN 10000000,BIN 11100000,BIN
11100000,BIN 11000000,0,0,0
 310 DATA 0,BIN 10000001,BIN 11100111,255,255,BIN 111
00111,BIN 10000001,0
 320 DATA 127,32,74,46,77,46,32,77,65,89,79,82,65,76,
32,83,69,82,82,65,78,79
 330 POKE 23658,8
```



```
340 RESTORE 10
350 FOR N=1 TO LEN G$
360 FOR K=0 TO 7
370 READ A
380 POKE USR G$(N)+K,A
390 NEXT K
400 NEXT N
410 INK 9: CLS
420 REM DIBUJO BOMBA
430 CIRCLE 51,100,40
440 PRINT  INK 6;AT 7,4;"AB CD"
450 PRINT  INK 6;AT 8,4;" FGH "
460 PRINT  INK 6;AT 9,4;" JKL "
470 PRINT  INK 6;AT 10,4;"MNOPQ"
480 PRINT  INK 6;AT 11,4;"RSTUV"
490 PLOT 46,141: DRAW 0,10
500 DRAW 12,0: DRAW 0,-10
510 FOR X=50 TO 53
520 PLOT X,151
530 DRAW 0,23
540 NEXT X
550 FOR X=48 TO 50
560 PLOT X,142: DRAW 0,7
570 NEXT X
580 FOR R=1.3 TO 1.5 STEP .05
590 PLOT 40,65
600 DRAW 0,70,-PI/R
610 NEXT R
620 LET FALLO=0
630 IF SW=1 THEN  RETURN
640 REM ELECCION PALABRA
650 LET PAL=INT (RND*50)+1
660 IF D(PAL) THEN  GO TO 650
670 LET D(PAL)=1
680 RESTORE 1140
690 FOR N=1 TO PAL
700 READ A$
710 NEXT N
720 DIM N$(LEN A$): GO SUB 930
730 REM INTRODUCCION INTENTOS
740 PRINT  PAPER 6;AT 21,0;"          INTRODUCE LETRA
    "
750 LET K$=INKEY$
760 IF CODE K$<65 OR CODE K$>90 THEN  GO TO 750
770 BEEP .2,40
780 PRINT AT 21,0; PAPER 0;"
    "
790 REM JUEGO
800 LET SWCO=0
810 FOR N=1 TO LEN A$
820 IF A$(N)=K$ THEN  PRINT  INK 6;AT 10,14+N;K$: LE
T SWCO=1: LET N$(N)=A$(N)
830 NEXT N
840 IF A$=N$ THEN  GO TO 1010
850 IF SWCO THEN  GO TO 730
860 PRINT AT 19,(FALLO+3)*2;K$
870 LET FALLO=FALLO+1
880 RANDOMIZE
890 IF FALLO=8 THEN  GO SUB 980: GO TO 1200
900 GO SUB 980
910 GO TO 730
920 REM  RAYITAS
930 FOR R=1 TO LEN A$
940 PRINT  INK 4;AT 10,14+R;"-"
950 NEXT R
960 RETURN
970 REM ANIMACION
980 PRINT  INK 4;AT 0,31-FALLO*3;"III"
990 PRINT  OVER 1; BRIGHT 1; FLASH 1; PAPER 6; INK 2
;AT 0,30-FALLO*3;"8"
1000 RETURN
1010 REM  PALABRA ACERTADA
1020 FOR N=0 TO 31
1030 PRINT  PAPER 0;AT 19,N;" "
1040 NEXT N
1050 LET P$="MUY BIEN MUCHACHO!"
1060 LET YY=130: LET XS=1: LET YS=2
1070 BRIGHT 1: INK 4
1080 GO SUB 1550: INK 9: BRIGHT 0
1090 FOR N=1 TO 50
1100 BORDER 3:::::: BORDER 4: BORDER 6:
1110 BEEP .01,40
1120 NEXT N
1130 PAPER 0: BORDER 0: CLS : GO TO 430
1140 REM B.DATOS (50 PALABRAS)
1150 DATA "TRANSFORMADOR","BUTACON","CARPETA","TELEFO
NO","SOFA","CIGARRO","SPECTRUM","IMPRESORA","MICROSCO
PIO","CARTERA"
1160 DATA "LIBRO","INTERFACE","PESCADO","LEOPARDO","C
OMPAS","TELEVISOR","COCINA","LAGARTO","ZAPATILLA","TE
CNOLOGIA"
1170 DATA "LEXICO","PTERODACTILO","LASER","CENICERO",
"CUARTILLA","GATITA","ANALISTA","FERNANDO","INGELEK",
"TELEX"
1180 DATA "VISITANTE","VULGARIDAD","ZANAHORIA","PSIQU
IATRIA","MAYORDOMO","MAYORAL","FOTOGRABADO","MILITAR"
,"AMETRALLADORA","ESTADISTICA"
1190 DATA "DESLIZAR","ABECEDARIO","COMBUSTIBLE","CABE
CERA","AXIOMATICO","XILOFONO","VENTRICULO","NITROGLIC
ERINA","PIRAMIDE","LEPIDOPTERO"
1200 REM  FIN JUEGO
1210 FOR N=31 TO 7 STEP -1
1220 PRINT  PAPER 0;AT 0,N;" "
1230 NEXT N
1240 PRINT  PAPER 2; INK 6; FLASH 1; BRIGHT 1; OVER 1
;AT 1,6;" ";AT 2,6;"
    "
1250 FOR N=1 TO 20
1260 BEEP .05,40: BEEP .04,30
1270 NEXT N
1280 FOR Y=175 TO 159 STEP -1
1290 PLOT 50,Y
1300 DRAW  FLASH 0; BRIGHT 0; OVER 1; PAPER 0; INK 0;
5,0
1310 NEXT Y
1320 FOR C=7 TO 0 STEP -1
1330 FOR N=1 TO 12
1340 PRINT  OVER 1; PAPER C; INK 9;AT N+2,1;"
    "
1350 NEXT N
1360 NEXT C
1370 FOR N=1 TO 12
1380 PRINT  PAPER 0;AT N+2,1;"          "
1390 NEXT N: CLS
1400 LET P$="BOOM!!"
1410 LET XS=5: LET YS=8: LET YY=30
1420 INK 3
1430 GO SUB 1550
1440 LET P$="LA PALABRA ES..."
1450 LET XS=2: LET YS=3: LET YY=100
1460 INK 7
1470 GO SUB 1550
1480 LET P$=A$
1490 LET XS=2: LET YS=2: LET YY=130
1500 INK 6
1510 GO SUB 1550
1520 IF INKEY$="" THEN  GO TO 1520
1530 INK 9: CLS : GO TO 430
1540 REM SBR. CARACT. GIGANTES
1550 LET xx=(256-8*xs*LEN p$)/2
1560 LET i=23306
1570 POKE i,xx: POKE i+1,yy: POKE i+2,xs: POKE i+3,ys
: POKE i+4,8
1580 LET i=i+4: LET u=LEN p$
1590 FOR q=1 TO u
1600 POKE i+q,CODE p$(q)
1610 NEXT q
1620 POKE i+u+1,255
1630 LET u=USR 32256
1640 RETURN
1650 REM  AUTO-EJEC. OBLIGATORIA
1660 CLEAR 32255
1670 LOAD ""CODE
1680 CLS
1690 LET SW=1: GO SUB 50
1700 LET SW=0
1710 LET P$="LA"
1720 LET XX=100: LET XS=2: LET YS=2: LET YY=30
1730 GO SUB 1560
1740 LET XS=4: LET YS=11: LET YY=50
1750 LET K$="BOMBA"
1760 FOR N=1 TO 5
1770 LET XX=N*32+65
1780 LET P$=K$(N)
1790 INK N+1
1800 GO SUB 1560
1810 NEXT N
1820 LET P$=""
1830 FOR N=1 TO 22
1840 READ ABDUL
1850 LET P$=P$+CHR$ ABDUL
1860 NEXT N
1870 PRINT  INK 9;AT 20,1;P$
1880 FOR N=1 TO 200
1890 NEXT N
1900 BEEP .5,30
1910 CLS
1920 INK 9
1930 GO TO 430
```

33	15	91	126	35	34	0	91
111	60	200	38	0	41	41	41
237	75	54	92	9	62	8	50
4	91	58	11	91	50	9	91
58	10	91	50	8	91	62	9
50	5	91	126	35	34	2	91
7	50	6	91	58	5	91	61
32	50	58	4	91	61	32	24
58	14	91	71	58	12	91	79
58	10	91	129	5	32	252	50
10	91	42	0	91	195	3	126
50	4	91	58	13	91	71	58
9	91	128	50	9	91	42	2
91	195	32	126	50	5	91	58
12	91	71	58	9	91	50	7
91	58	13	91	79	197	205	164
126	193	58	7	91	60	50	7
91	13	32	241	58	8	91	60
50	8	91	5	32	221	58	6
91	195	48	126	128	64	32	16
8	4	2	1	58	142	92	238
255	71	58	141	92	160	71	58
8	91	230	248	111	58	7	91
254	192	208	31	31	31	230	31
103	203	28	203	29	203	28	203
29	203	28	203	29	62	88	180
103	58	142	92	166	176	119	58
7	91	71	230	7	246	64	103
120	31	31	31	230	24	180	103
120	23	23	230	224	111	58	8
91	71	31	31	31	230	31	181
111	235	33	156	126	120	230	7
79	6	0	9	70	26	33	6
91	203	70	40	3	176	18	201
47	176	47	18	201			