INTRODUCCIÓN A LA PROGRAMACIÓN EN BASIC

INDICE

0. Introducción A La Programación En BASIC.

PRÓLOGO.

Vamos a ver una introducción a la programación de ordenadores. Para ello, vamos a centrarnos en el lenguaje BASIC. Es un lenguaje que durante mucho tiempo se consideró como muy apropiado para comenzar en la programación de ordenadores, y que posteriormente fue desbancado para este propósito por otros lenguajes más estructurados como Pascal. Hoy en día existen versiones de Basic lo suficientemente estructuradas como para resultar adecuadas para aprender correctamente los principios básicos, y versiones lo suficientemente potentes como para realizar aplicaciones "serias" en entornos como Windows.

Nosotros comenzaremos centrándonos en los Basic "convencionales", viendo cómo se realizan programas sencillos en modo texto, y después iremos aplicándolo a otros Basic "más modernos".

Para seguir el curso, podrás usar un ordenador clásico (Amstrad CPC, MSX, Commodore 64), un emulador de alguno de ellos, un PC antiguo con MS-DOS, o también podrás instalar algún intérprete de BASIC gratuito (que te recomendaremos más adelante) en un equipo moderno con Windows o Linux.

1. Nociones Elementales Sobre Los Programas Y BASIC

Un **programa** es un conjunto de órdenes para un ordenador. Estas órdenes se deben dar en un cierto lenguaje. Lo habitual es emplear lenguajes cercanos al lenguaje humano (normalmente parecidos al inglés), y emplear una cierta aplicación que convierte esas órdenes al lenguaje que realmente entiende el ordenador. Estos **"traductores"** reciben el nombre de compiladores o de intérpretes, según cómo se realice la traducción (no entraremos en más detalles por ahora).

Veremos el manejo de varios **intérpretes** distintos de Basic. En muchos de ellos, el hecho de teclear una orden y pulsar la tecla de introducción (Intro o Enter) hace que esa orden se ponga en funcionamiento inmediatamente; si queremos teclear un programa completo, para que quede almacenado en la memoria, y se ponga en funcionamiento más tarde, deberemos preceder cada línea por un número correlativo.

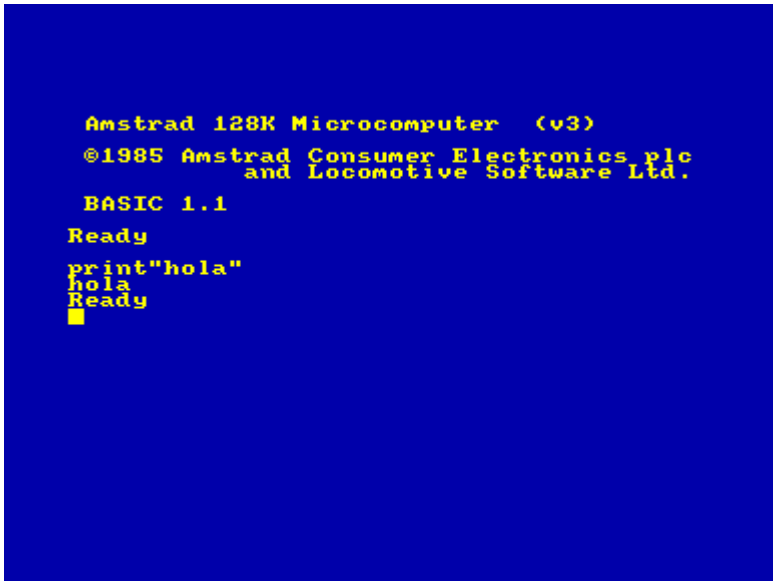
Así, en la mayoría de los **Basic "antiguos"** , (como el que incluían los ordenadores Sinclair ZX Spectrum, MSX, Amstrad CPC, Commodore 64, etc., o el GwBasic de las primeras versiones de MsDos, o el de muchas calculadoras programables), si tecleamos

```
PRINT "HOLA"
```

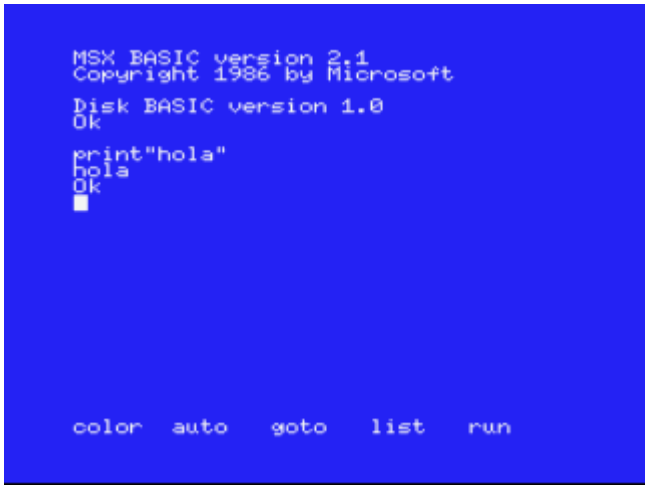
(y pulsamos la tecla Intro) el ordenador hará caso a esta orden (la " **ejecutará**") y responderá mostrando el mensaje HOLA en la pantalla.

Nota: PRINT es la orden que escribe un cierto texto en pantalla. Profundizaremos en ella en el siguiente apartado. En adelante daré por entendido que deberemos pulsar la tecla Intro (o su equivalente en el ordenador que empleemos) después de cada orden, y no insistiré más en ello.

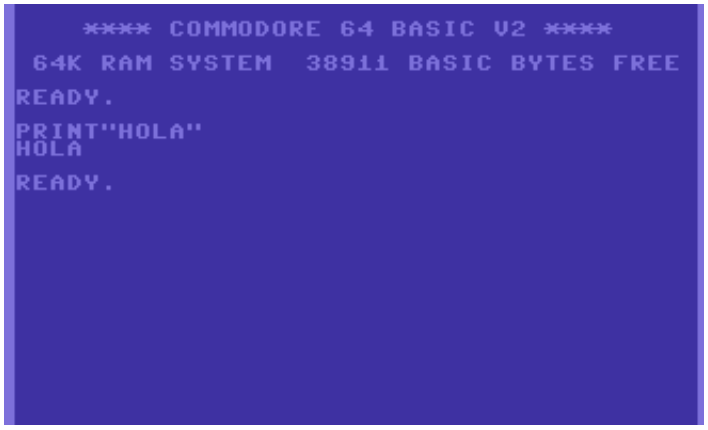
Éste sería el resultado si tecleamos PRINT "HOLA" y luego pulsamos Intro, usando un ordenador Amstrad CPC, de mediados de los años 80.



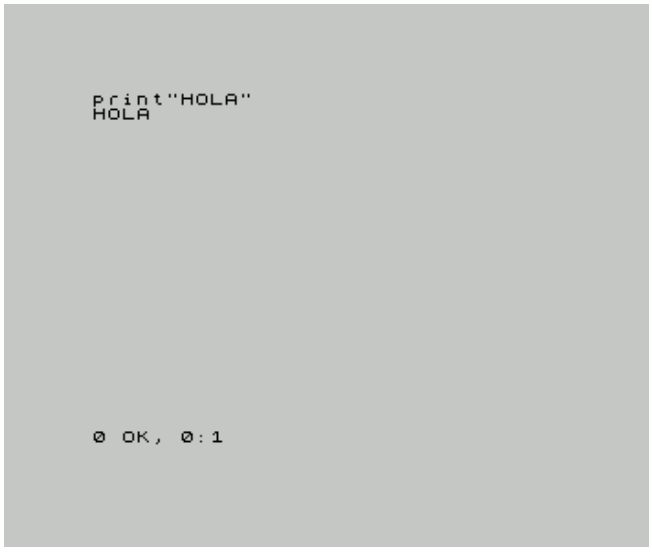
Y éste usando un equipo MSX de la misma época



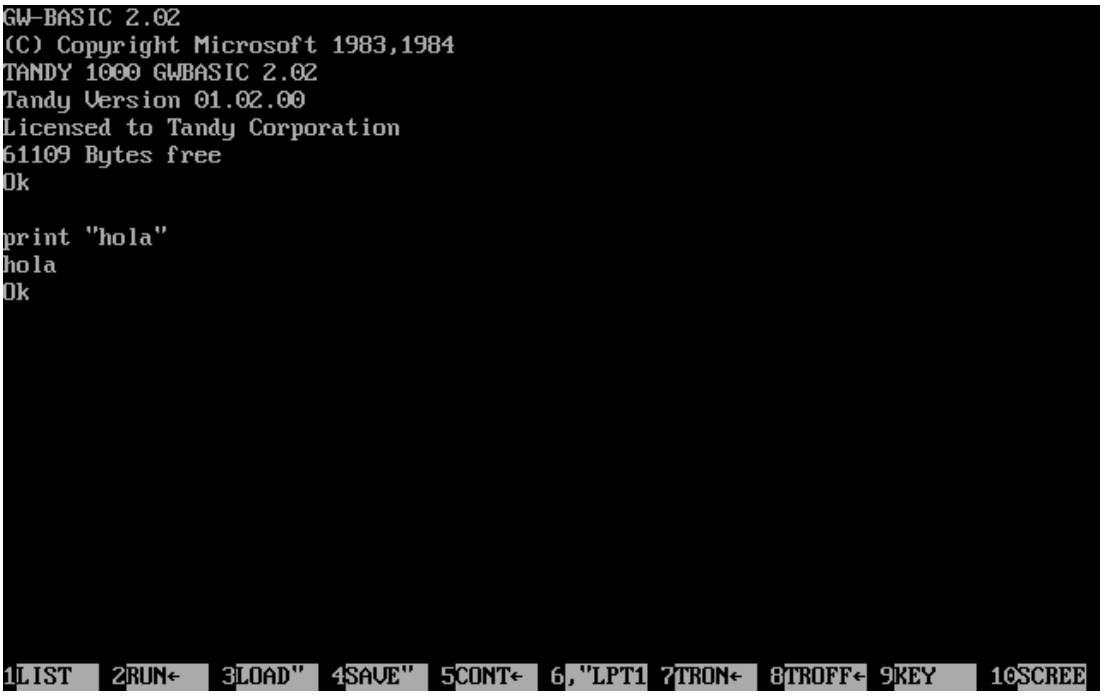
Y el mismo resultado con un Commodore 64:



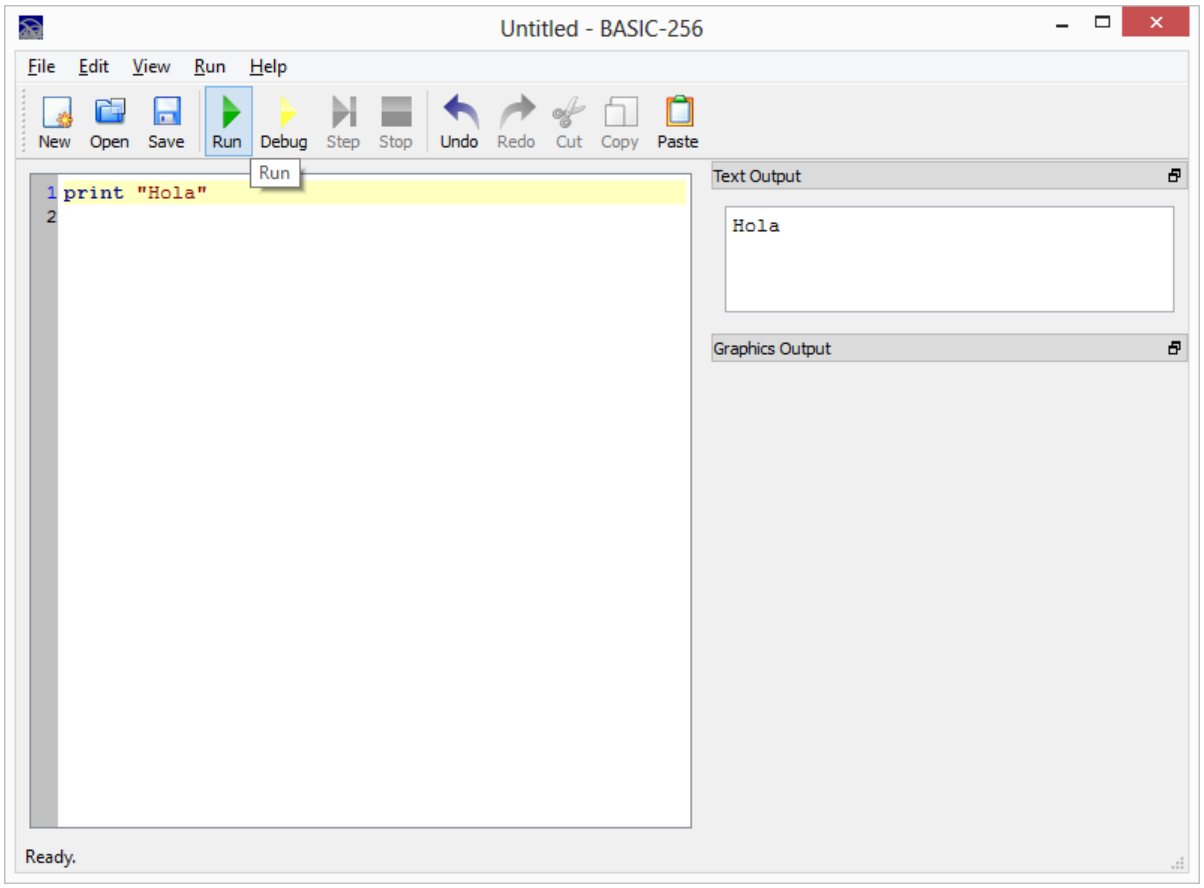
La gama Sinclair ZX Spectrum de principios y mediados de los 80 pueden no ser los mejores ordenadores para comenzar, especialmente si usas un emulador, porque cada tecla tenía un comportamiento diferente según el momento. Por ejemplo: al comenzar, las teclas se comportaban en "modo de orden directa", y con una sola pulsación de la tecla P aparecía la orden PRINT escrita en pantalla. Esto hace más complicado su uso desde un emulador, si no se tiene al lado una imagen que recuerde cómo era el teclado original. Aun así, en la gama 128 K se podía trabajar de una forma un poco más natural. Ésta sería nuestra primera orden ejecutada en un Spectrum 128k:



Un PC con MS-DOS y el intérprete llamado GwBasic mostraría una apariencia similar a los anteriores:



¿Sólo podrás aprender BASIC si tienes un ordenador clásico o un emulador? Por supuesto que no. Si tienes un PC moderno con Windows o Linux, puedes descargar gratuitamente desde Internet un intérprete de BASIC llamado Basic256, que te permitirá probar los ejercicios de este curso. Si usas este entorno, deberás teclear tus órdenes y luego pulsar el botón "Run". El resultado aparecerá en el panel derecho:



Nota: La gran mayoría de versiones del lenguaje BASIC no distinguen entre órdenes en mayúsculas y en minúsculas, de modo que **PRINT** será lo mismo que **print** o **Print.**

En un ordenador clásico, si queremos teclear un programa que primero escriba HOLA y luego ADIOS, y queremos que este programa se ponga en funcionamiento más adelante, cuando nosotros se lo pidamos, en vez de lanzarse cada orden inmediatamente, teclearíamos

```
1 PRINT "HOLA"
2 PRINT "ADIOS"
```

(Fuente "0101.bas". Probado en GwBasic 2.02, Locomotive Basic 1.0 de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1, QBasic 1.1)

Estas dos órdenes precedidas por un número quedarán almacenadas. Cuando queremos que se pongan en funcionamiento, deberemos teclear la orden **RUN**, así:

```
RUN
```

El ordenador las analizará en orden numérico, primero la 1 y después la 2, por lo que mostraría la palabra HOLA, y bajo esta, la palabra ADIOS.

Puede ocurrir que, después de haber tecleado varias órdenes, nuestro programa inicial ya no esté visible en la pantalla. Para que el ordenador nos lo vuelva a mostrar, se emplea la orden **LIST**:

```
LIST
```

Un detalle sobre la **numeración de las líneas**: lo habitual en estos Basic "antiguos" era no numerar las líneas de 1 en 1, sino con una separación mayor, habitualmente de 10 en 10, de modo que nuestro primero programa quedaría

```
10 PRINT "HOLA"
20 PRINT "ADIOS"
```

(Fuente "0102.bas". Probado en GwBasic 2.02, Locomotive Basic, Commodore 64 Basic V2, Msx Basic 2.1)

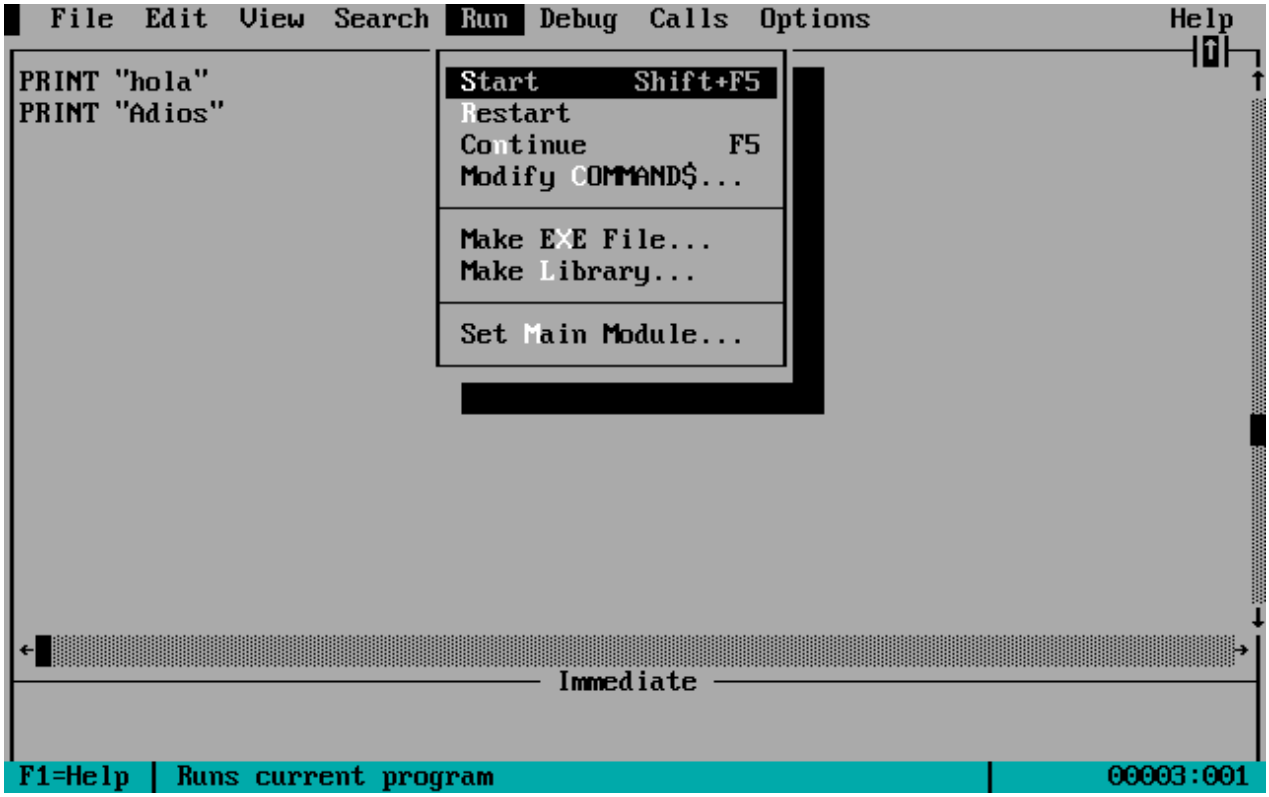
esto permite que podamos insertar nuevas líneas entre las ya existentes, utilizando un número intermedio. Por ejemplo, la línea 15 se "ejecutaría " después que la 10 y antes que la 20. A su vez, si añadimos una línea 12, ésta se pondría en funcionamiento después de la 10 y antes que la 15.

En los **BASIC "modernos"**, como el **Basic256** que hemos mencionado antes, o el **QBasic** que incorporaban las últimas versiones de MsDos, o el Turbo Basic de Borland, tendremos un editor en el que iremos tecleando las órdenes una tras otra, sin necesidad de números de línea (aunque algunos los permiten, pero no son imprescindibles):

```
PRINT "HOLA"
PRINT "ADIOS"
```

(Fuente "0103.bas". Probado en Basic256)

Para **poner el programa en funcionamiento**, deberemos hacer clic en cierto icono, como el "Run" de Basic256, o pulsar cierta combinación de teclas, que en el caso de QBasic es Mayúsculas+F5 y en el caso de Turbo Basic es Alt+X:



Algunos BASIC antiguos obligaban a indicar el **final del programa** con la palabra **END** (en la mayoría no es necesario: se sobreentiende que el programa acaba cuando no existan más líneas, de modo que esta orden sería opcional). En ese caso, nuestro programa quedaría así:

```
10 PRINT "HOLA"
20 PRINT "ADIOS"
30 END
```

(Fuente "0104.bas". Probado en GwBasic 2.02, Locomotive Basic, Commodore 64 Basic V2, Msx Basic 2.1)

¿Cómo **conseguir algún intérprete** de BASIC, en el que poder probar los ejemplos? Los recomendados son QBasic y Basic256. Aquí tienes detalles de cómo conseguir éstos y otros:

- Si tenemos un **ordenador tipo PC**, equipado con Windows, Basic256 es una de las mejores opciones para empezar. Su sintaxis se separa un poco del estándar, pero, a cambio, el entorno es bastante cómodo. Existe incluso una versión de Basic256 para tablets y smartphones con sistema operativo Android. Para PC más antiguos (o, mediante emuladores, también para ordenadores modernos), también hay otras alternativas, según el sistema operativo que usemos:
    - En las versiones antiguas de MsDos (hasta la 4.0, creo recordar) se incluía un intérprete "antiguo" de Basic llamado GwBasic. Para utilizarlo, debería bastar con teclear "gwbasic ", y para salir, habrá que teclear "exit". En otras versiones distribuidas por IBM (PcDos), puede que el intérprete de BASIC se llame "basica". Si usas un PC moderno, puedes usar esas versiones de BASIC con el emulador PCE, en una descarga directa, junto con PcDOS 3.30 o bien puedes probar Pc-Basic, un emulador de GwBasic, que está disponible para Windows, Mac, Linux, Android y otros.
    - En las versiones más recientes de MsDos (5.0 a 6.22), se incluía un intérprete "mejorado ", llamado QBasic. Para entrar, habrá que teclear "qbasic" , y la opción de Salir está en el menú Archivo. Puedes probarlo con el emulador DosBox y descomprimiendo este fichero ZIP que contiene QBasic, para dejarlo dentro de la carpeta VirtualHD.
    - QBasic también está disponible en Windows 95 y Windows 98, pero no se instala automáticamente. Deberíamos instalarlo "a mano", copiando los ficheros QBASIC.EXE y QBASIC.HLP del CdRom original de Windows (debería estar en la carpeta OLDMSDOS; que a su vez estará en TOOLS) a la carpeta WINDOWS\COMMAND del disco duro). Para utilizarlo, deberíamos entrar al modo MsDos (Inicio / Programas / MS-DOS) y teclear QBasic. En Windows modernos de 64 bits no funcionará y habrá que recurrir al emulador DosBox o alternativas similares.
    - QB64 es un clon de QBasic, capaz de funcionar en equipos de 64 bits, y para el que hay versiones para Windows, Linux y MacOS X.

- Si tienes algún **ordenador "antiguo"** (de los años 80), lo normal será que nada más encenderlo aparezca directamente el intérprete de BASIC. Es el caso de ordenadores como la gama Sinclair ZX (modelos 80, 81 y Spectrum), la gama Amstrad CPC (modelos 464, 472, 664, 6128), los que seguían el estándar MSX (había modelos realizados por varias casas, como Philips, Toshiba, Sanyo, Sony o Spectravideo), varios Commodore (64, 16, Vic-20), etc.
- Muchas **calculadoras programables**, como las PB100, FX850 y FX880 de Casio incorporan también el lenguaje Basic.
- Además, en casi cualquier ordenador moderno, tenemos la opción de instalar un **emulador** de un ordenador "antiguo" (como los MSX o los Amstrad CPC). Cada vez que entremos al emulador, será como si estuviésemos sentados delante de alguno de aquellos ordenadores. Eso sí, **no recomiendo** usar emuladores del Sinclair ZX Spectrum a quien no haya manejado este ordenador anteriormente, porque el manejo de teclado puede resultar bastante desconcertante. Los Commodore 64 y Vic-20 tenían un teclado mucho más estándar, pero su versión de Basic tenía muchas carencias, por lo que tampoco son los más recomendables. Algunos emuladores de estos sistemas son:
    - Para emular un Amstrad CPC desde Windows puedes usar WinApe, que permite "copiar y pegar" desde el ordenador anfitrión .
    - Para un MSX desde Windows, BlueMSX.
    - Para un Commodore tienes VICE, que permite "copiar y pegar" desde el ordenador anfitrión (el programa a copiar deberá estar en minúsculas).

Ejercicio propuesto 1.1: *Consigue una versión de BASIC para tu ordenador y crea un programa que te salude por tu nombre (deberá escribir en pantalla algo como "Hola, Nacho")*

2. Escribir En Pantalla; Operaciones Matemáticas

Ya hemos comentado la idea básica sobre cómo se escribe en pantalla utilizando el lenguaje Basic: emplearemos la orden PRINT, y entre comillas indicaremos el texto que queremos que aparezca:

```
PRINT "HOLA"
```

(Recuerda que, en los equipos antiguos, esta será una orden directa, que se pondrá en marcha inmediatamente).

Nota: A partir de ahora, nuestros programas usarán colores como éstos, para ayudar a distinguir los números, las órdenes y los textos.

Si después de esta orden escribimos otra orden PRINT, cada uno de los mensajes **se escribirá en una línea distinta**. Esto es porque tras cada orden PRINT, el cursor (la posición de la pantalla en la que se seguirá escribiendo) avanza hasta el comienzo de la siguiente línea.

Si nos interesa que los dos mensajes **aparezcan en una misma línea** , justo uno detrás del otro, podemos conseguirlo escribiendo un **punto y coma** (;) después de la primera orden PRINT. Esto hace que el cursor no avance a la siguiente línea.

```
10 PRINT "HOLA";
20 PRINT "ADIOS"
```

(Fuente "0201.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

(Recuerda que deberás teclear RUN para probar un programa con el anterior si empleas un equipo antiguo).

La mayoría de las versiones de BASIC permiten que los dos mensajes aparezcan en una misma línea, pero no juntos, sino **separados** un cierto número de espacios, lo que conseguimos utilizando una **coma**(,) en vez de un punto y coma, así:

```
10 PRINT "HOLA",
20 PRINT "ADIOS"
```

(Fuente "0202.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

Estos dos textos se pueden imprimir en una única línea PRINT, así:

```
10 PRINT "HOLA"; "ADIOS"
```

(Fuente "0203.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

(en este caso el ordenador escribiría HOLAADIOS) o así

```
10 PRINT "HOLA", "ADIOS"
```

(Fuente "0204.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

(en este caso escribiría HOLA ADIOS).

Nota 1: Basic256 permite terminar una línea con "punto y coma", para seguir más adelante escribiendo en la misma línea, pero no permite la "coma" y tampoco permite continuar la orden tras el punto y coma, de modo que no sería válida la orden: PRINT "HOLA"; "ADIOS"

Nota 2: algunas versiones de BASIC permiten indicar cuántos espacios saltará esta coma. Por ejemplo, en el BASIC de los Amstrad CPC existía una orden **ZONE** con esta misión, que se usaría así:

```
5 ZONE 12
```

de modo que el programa completo sería

```
5 ZONE 12
10 PRINT "HOLA", "ADIOS"
```

(Fuente "0205.bas". Probado en Locomotive Basic de Amstrad CPC)

Podemos **escribir números** (y entonces no harán falta las comillas), e incluso operaciones entre ellos:

```
10 PRINT 23
20 PRINT 44 + 5;
30 PRINT 50-9
```

(Fuente "0206.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2)

La salida de este programa sería

```
23
49 41
```

(**Nota**: en este caso, para muchas versiones de BASIC, los números no aparecen totalmente juntos, como ocurría con los textos, sino separados por un espacio en blanco, que es donde aparecería el signo - si los números fueran negativos).

Las **operaciones** matemáticas se representan con los siguientes símbolos:

```
+ Suma
- Resta y negación.
* Multiplicación
/ División
^ Exponenciación (elevado a: el acento circunflejo o la flecha hacia arriba)
```

Y un ejemplo de su uso sería

```
10 PRINT "Ejemplo de operaciones, con los numeros 5 y 2"
20 PRINT "Suma: "; 5 + 2
30 PRINT "Diferencia: "; 5 - 2
40 PRINT "Producto: "; 5 * 2
50 PRINT "Division: "; 5 / 2
60 PRINT "Potencia: "; 5 ^ 2
```

(Fuente "0207.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

que tendría como resultado:

```
Ejemplo de operaciones, con los numeros 5 y 2
Suma: 7
Diferencia: 3
Producto: 10
Division: 2.5
Potencia: 25
```

Una versión de esta fuente para Basic256, que no permite números de línea ni separar varias expresiones con "punto y coma" sería:

```
PRINT "Ejemplo de operaciones, con los numeros 5 y 2"
PRINT "Suma: ";
PRINT 5 + 2
PRINT "Diferencia: ";
PRINT 5 - 2
PRINT "Producto: ";
PRINT 5 * 2
PRINT "Division: ";
PRINT 5 / 2
PRINT "Potencia: ";
PRINT 5 ^ 2
```

(Fuente "0208.bas". Probado en Basic256

En muchas las versiones de Basic, la orden PRINT se puede abreviar por una **interrogación** (?), de modo que las dos siguientes líneas harían exactamente lo mismo:

```
10 PRINT "HOLA"
20 ? "HOLA"
```

(Fuente "0209.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Msx Basic 2.1)

Es más, algunas versiones de Basic convertirían automáticamente la segunda línea a ésta:

```
20 PRINT "HOLA"
```

Más adelante veremos mayores detalles sobre cómo mejorar la salida en pantalla utilizando PRINT.

Ejercicio propuesto 2.1: *Crea un programa en BASIC que te diga la suma de 234 y 731.*

Ejercicio propuesto 2.2: *Crea un programa en BASIC que muestre la diferencia (resta) entre 731 y 234.*

Ejercicio propuesto 2.3: *Crea un programa en BASIC que te diga el resultado de multiplicar 125 por 32.*

Ejercicio propuesto 2.4: *Crea un programa en BASIC que calcule y muestre el cociente (división) de 800 entre 14.*

Ejercicio propuesto 2.5: *Crea un programa en BASIC que calcule la cuarta potencia de 5.*

Ejercicio propuesto 2.6: *¿Cuál será el resultado de 5 + 3 x 4? Crea un programa en BASIC que lo calcule y comprueba si has acertado.*

3. Introducción A Las Variables

Habíamos visto cómo escribir un texto que habíamos fijado nosotros mismos desde el programa. Pero lo habitual no es escribir valores fijos, sino el resultado de una cierta operación, un valor introducido por el usuario, etc. Para eso se emplean las llamadas "variables": una **variable** es un espacio de memoria en el que podremos guardar un cierto dato y al que podremos asignar un nombre. Este dato lo podremos modificar cuando queramos. Por ejemplo, podemos reservar un espacio de variable y llamarlo "X" y guardar en él un cierto valor, así

```
10 LET X = 2
20 PRINT "X VALE:"
30 PRINT X
```

(Fuente "0301.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

Como se ve, la orden **LET** es la que permite dar un valor a una cierta variable. Para mostrar en pantalla el valor de una variable, utilizamos la orden PRINT, pero indicando el nombre de la variable sin comillas. El resultado del programa anterior sería:

```
X VALE:
2
```

Más adelante podemos **modificar** el valor de X nuevamente:

```
10 LET X = 2
20 PRINT "X VALE:" ; X
30 LET X = 5
40 PRINT "AHORA X VALE:" ; X
```

(Fuente "0302.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

El resultado de este programa sería:

```
X VALE: 2
AHORA X VALE: 5
```

Aquí hay otra novedad: hemos incluido en una misma orden PRINT un texto entre comillas y una variable (sin comillas), separados por **punto y coma**, para que se escriban en la misma línea.

La orden LET **no es obligatoria** en la mayoría de las versiones del lenguaje Basic, de modo que muchos Basic permitirían que el programa anterior se escribiese así:

```
10 X = 2
20 PRINT "X VALE:" ; X
30 X = 5
40 PRINT "AHORA X VALE:" ; X
```

(Fuente "0303.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2)

Si queremos que sea el **usuario** el que introduzca un valor para una variable, se usa la orden **INPUT**, así:

```
10 INPUT X
20 PRINT "EL VALOR TECLEADO ES: " ; X
30 PRINT "SU CUADRADO ES:" ; X*X
```

(Fuente "0304.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

La orden INPUT mostrará una interrogación para avisarnos de que está esperando un valor. Por ejemplo, si el usuario introduce el valor 3, en pantalla aparecería lo siguiente:

```
?3
EL VALOR TECLEADO ES: 3
SU CUADRADO ES: 9
```

También podemos indicar en la orden INPUT un texto entre comillas, para que aparezca delante de la interrogación (no en todas las versiones de Basic, pero sí en la mayoría):

```
10 INPUT "QUE NUMERO DESEA ELEVAR AL CUADRADO"; X
20 PRINT "EL VALOR TECLEADO ES: " ; X
30 PRINT "SU CUADRADO ES:" ; X*X
```

(Fuente "0305.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

En el resto de versiones, la alternativa amigable será usar un PRINT para avisar al usuario y luego el INPUT para leer datos:

```
10 PRINT "QUE NUMERO DESEA ELEVAR AL CUADRADO";
20 INPUT X
30 PRINT "EL VALOR TECLEADO ES: " ; X
40 PRINT "SU CUADRADO ES:" ; X*X
```

(Fuente "0306.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Msx Basic 2.1)

En Basic256, el texto de INPUT se debe separar de la variable con una "coma", no con "punto y coma", y la interrogación no aparecerá automáticamente, así que el programa quedaría así:

```
INPUT "QUE NUMERO DESEA ELEVAR AL CUADRADO? ", X
PRINT "EL VALOR TECLEADO ES: " ;
PRINT X
PRINT "SU CUADRADO ES: " ;
PRINT X*X
```

(Fuente "0307.bas". Probado en Basic256)

Ejercicio propuesto 3.1: *Crea un programa que te pida dos números y muestre su suma*

Hasta ahora sólo hemos usado una variable llamada X, pero este no es un nombre demasiado claro. Normalmente nos interesará tener **nombres de variables** más significativos. Puede ocurrir que alguna versión de Basic antigua sólo permita nombres de variables formados por una única letra, pero lo habitual es que permitan nombres bastante largos:

```
10 PRIMERAVARIABLE = 20
20 SEGUNDAVARIABLE = 55
30 RESULTADO = PRIMERAVARIABLE * SEGUNDAVARIABLE
40 PRINT "RESULTADO VALE:" ; RESULTADO
```

(Fuente "0308.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

En la gran mayoría de versiones del lenguaje BASIC, serán válidos nombres de variables que empiecen por una letra y después contengan letras o números (y en algún caso, algún otro símbolo, como el signo de subrayado). No podrán contener espacios, caracteres internacionales (letras acentuadas, eñes, etc.), ni casi ningún signo de puntuación (puntos, comas, etc.). Algunos símbolos especiales podrán aparecer al final del nombre de una variable, y tendrán un cierto significado, pero eso lo veremos un poco más adelante...

Eso sí, existen algunas versiones antiguas de BASIC que permiten **nombres de variables largos**, pero desprecian las letras que aparecen a partir de una cierta posición. Por ejemplo, si un intérprete de BASIC sólo considera las ocho primeras letras de cada variable, considerará que una variable llamada VARIABLETEMPORAL1 y otra llamada VARIABLETEMPORAL2 son la misma. Es algo que deberíamos comprobar con la versión de Basic que usemos, porque puede dar lugar a errores muy difíciles de detectar.

Además, algunas versiones de BASIC **no sólo usan los espacios como separadores** sino que también miran dónde termina cada palabra reservada. En estos BASIC (como el de los Commodore 64 y las calculadoras programables CASIO FX), las órdenes "X=5:PRINTX" escribirían un 5 en pantalla, mientras que en la mayoría de versiones de BASIC obtendríamos un mensaje de error porque el sistema no sabría qué es "PRINTX". Por el mismo motivo, en estas versiones de BASIC no podremos crear variables cuyo nombre comience como una palabra reservada: "PRINTER=5" no estaría creando una variable ER con el valor 5, sino que escribiría en pantalla si la variable ER tiene el valor 5 (hablaremos de comprobación de condiciones más adelante).

Otro problema que puede existir con las variables se debe al hecho de que en el lenguaje Basic no es necesario **declarar una variable** antes de usarla: no hace falta decir "voy a usar una variable llamada X", sino que el propio intérprete de BASIC, cada vez que ve una variable nueva, la acepta como válida y **le da el valor 0**. ¿Qué problemas puede provocar esto? Vamos a verlo con un ejemplo:

```
10 PRIMERAVARIABLE = 20
20 SEGUNDAVARIABLE = 55
30 RESULTADO = PRIMERAVARIABLE * SEGUDNAVARIABLE
40 PRINT "RESULTADO VALE:" ; RESULTADO
```

(Fuente "0309.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Msx Basic 2.1)

Me temo que el resultado que obtenemos no es el que queríamos obtener: 20*55 = 1100, pero sin embargo nuestro programa está respondiendo RESULTADO VALE: 0 ¿Por qué? El fallo está en la línea 30: no hemos escrito SEGUNDAVARIABLE, sino SEGUDNAVARIABLE, hemos cambiado de sitio las letras N y D. En otros lenguajes, esto haría que recibiésemos una aviso de que la variable llamada SEGUDNAVARIABLE no se ha declarado previamente, pero en Basic no ocurre así: se da por sentado que hemos escrito correctamente la variable, y se le da el valor 0. Por eso el resultado de nuestro programa es 22*0 = 0, correcto para el intérprete de Basic... pero posiblemente incorrecto para nosotros.

Esto supone que haya que teclear con **MUCHO CUIDADO** y que aun así se deba PROBAR todos los programas con varios datos, para los que sepamos el resultado que se deba obtener. En cuanto notemos algo raro, habrá que revisar el programa.

Algún BASIC muy moderno (como QBasic o Visual Basic) permitirá evitar este tipo de problemas, pero eso también lo veremos más adelante.

Las variables que hemos estado utilizando hasta ahora estaban almacenando números. Concretamente, almacenaban números enteros (sin decimales). Esto es lo que permitirá cualquier versión del lenguaje BASIC. En algunas versiones podremos almacenar otros tipos de datos, como números reales (con decimales), de distintas precisiones, y ya veremos cómo indicar a nuestro intérprete de BASIC que se trata de números reales o de cualquier otro tipo de datos.

Pero existe un tipo de datos que podremos emplear en casi cualquier versión de BASIC, y que se utiliza mucho en la práctica: se trata de **cadenas de texto**, conjuntos de una o más letras que utilizaremos para escribir mensajes en pantalla.

Las variables que almacenen cadenas de texto deberán tener nombres terminados con el signo **$**:

```
10 LET NOMBRE$="JUAN PEDRO"
20 PRINT "HOLA " ; NOMBRE$
```

(Fuente "0310.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

Podemos "concatenar" dos cadenas de texto con el signo "+", obteniendo una nueva cadena de texto formada por las dos de partida:

```
10 LET SALUDO$="HOLA "
20 INPUT "INTRODUCE TU NOMBRE"; NOMBRE$
30 FRASE$ = SALUDO$ + NOMBRE$
40 PRINT FRASE$
```

(Fuente "0311.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

Así, si el usuario teclea ALBERTO cuando se le pide su nombre, el valor de FRASE$ (que se mostraría en pantalla en la línea 40) sería HOLA ALBERTO

Con Basic256, esta programa sería casi idéntico, salvo que el texto de INPUT se debe separar de la variable con una "coma":

```
SALUDO$="HOLA "
INPUT "INTRODUCE TU NOMBRE: ", NOMBRE$
FRASE$ = SALUDO$ + NOMBRE$
PRINT FRASE$
```

(Fuente "0312.bas". Probado en Basic256)

Ejercicio propuesto 3.2: *Crea un programa que te pida tu nombre (por ejemplo, "Juan") y tu edad (por ejemplo, 15) y diga que aparentas tener menos años: "Hola, Juan, pareces tener menos de 15 años".*

Ejercicio propuesto 3.3: *Crea un programa que te pida tu nombre, tu apellido, y escriba "Bienvenido, sr. " seguido por tu apellido, una coma y tu nombre.*

4. Los Comentarios

A veces nos puede interesar que en nuestro programa aparezcan comentarios, que a nosotros nos aclaren por qué hemos dado un cierto paso. Será muy útil cuando se trabaje en grupo, o incluso cuando nosotros mismos tengamos que revisar un programa un cierto tiempo después de haberlo creado: nos ahorrará tiempo de "descifrar " qué habíamos hecho y por qué.

Los **comentarios en BASIC** se indican con la palabra REM (abreviatura de REMARK, comentario, en inglés):

```
10 REM SIMPLEMENTE UN SALUDO
20 PRINT "HOLA"
```

(Fuente "0401.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

La gran mayoría de las versiones del lenguaje BASIC permitirán abreviar todavía más la palabra REM, poniendo un apóstrofe (') en su lugar:

```
10 ' SIMPLEMENTE UN SALUDO
20 PRINT "HOLA"
```

(Fuente "0402.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Msx Basic 2.1 - No se acepta en Commodore 64 Basic V2)

En el caso de Basic256, el símbolo para comentarios es una almohadilla:

```
# SIMPLEMENTE UN SALUDO
PRINT "HOLA"
```

(Fuente "0403.bas". Probado en Basic256)

Ejercicio propuesto 4.1: *Crea un programa que te pida dos números y muestre su producto. La primera línea debe ser un comentario que diga la misión del programa. La segunda línea debe ser un comentario con tu nombre.*

5. Varias Líneas En Una.

Hasta ahora hemos estado poniendo cada orden en una línea, pero esto no tiene por qué ser así: también podemos incluir **varias órdenes en una misma línea**, si las separamos con el símbolo de "dos puntos" (**:**)

```
10 INPUT "DAME TU NOMBRE"; NOMBRE$ : PRINT "HOLA, "; NOMBRE$
20 PRINT "HASTA LUEGO" : REM AQUI SE ACABA
```

(Fuente "0501.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

En el ejemplo anterior, hemos incluido una orden INPUT y una PRINT en la misma línea, y después una PRINT y una REM también en una misma línea.

Una observación: será **peligroso** hacer cosas como

```
20 REM AQUI SE ACABA : PRINT "HASTA LUEGO"
```

porque la mayoría de las versiones de Basic considerará como comentario todo lo que sigue a la orden REM, y en este caso eso incluye también a los dos puntos, a la orden PRINT y al texto entre comillas, de modo que la orden PRINT no llegaría a actuar.

Algunos BASIC más modernos que no usen números de línea pueden no permitir escribir varias órdenes en la misma línea. En el caso de Basic256, la documentación no menciona que se pueda utilizar "dos puntos" para separar órdenes, a pesar de lo que el siguiente programa se reconoce perfectamente... excepto en el caso de que sea una orden "REM" lo que haya tras los dos puntos (al menos en la versión 1.1.0 de Basic256):

```
REM Varias líneas en una
# Y con comentarios
INPUT "DAME TU NOMBRE ", NOMBRE$ : PRINT "HOLA, "; : PRINT NOMBRE$
PRINT "HASTA LUEGO": # Esta línea da ERROR si tiene comentario
```

(Fuente "0502.bas". Probado en Basic256)

Vamos a ver un ejemplo, que pida dos números al usuario y muestre su suma, pero ya con un poco más de corrección y aplicando algunas de las últimas cosas que hemos visto:

```
10 REM Sumar dos numeros que introduzca el usuario
20 PRINT "Introduce los dos numeros que quieres sumar"
30 INPUT "Primer numero"; x  : INPUT "Segundo numero"; y
40 PRINT "Su suma es "; x+y
```

(Fuente "0503.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

Ejercicio propuesto 5.1: *Crea un programa que te pida dos números n1 y n2 y muestre el resultado de 3*(n1+n2) y el de 3*n1 + 3*n2 (si todo es correcto, deberías obtener el mismo resultado en ambas operaciones) Las dos órdenes que escriben resultados deben estar en la misma línea.*

6. Los Bucles: FOR

Llamaremos "**bucle**" a una parte del programa que se repite un cierto número de veces. En BASIC, la forma más estándar de conseguirlo es con la orden **FOR**, que vamos a ver directamente con un ejemplo:

```
10 REM Escribimos un saludo 10 veces
20 FOR i = 1 TO 10
30 PRINT "Hola"
40 PRINT "Como estas?"
50 NEXT i
60 PRINT "Se acabaron las repeticiones"
```

(Fuente "0601.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

En español, este FOR se podría traducir como un "Desde", de modo que la línea 20 sería "desde que i valga 1 hasta 10", es decir estamos usando una variable auxiliar, que comenzará valiendo 1, y el trozo de programa se repite hasta que esa variable i valga 10, aumentando de 1 en 1.

La línea 50 indica el final de la parte que se repite, y hace que se pase al próximo (next) valor de la variable i. Es decir, hace que pase de 1 a 2 y que se vuelvan a repetir las líneas 30 y 40; en la siguiente pasada, i aumenta de 2 a 3, y se vuelven a repetir las líneas 30 y 40; así sucesivamente, hasta que i vale 10, momento en que dejan de repetirse las líneas 30 y 40.

El resultado de este programa será:

```
Hola
Como estas?
Hola
Como estas?
Hola
Como estas?
Hola
Como estas?
Hola
Como estas?
Hola
Como estas?
Hola
Como estas?
Hola
Como estas?
Hola
Como estas?
Hola
Como estas?
Hola
Como estas?
Hola
Como estas?
Se acabaron las repeticiones
```

Es muy frecuente emplear la orden FOR para contar, aprovechando eso de que la variable aumenta desde un valor que indicamos hasta el otro. Por ejemplo, podemos contar del 1 al 8 así:

```
10 REM Escribimos del 1 al 8
20 FOR n = 1 TO 8
30 PRINT n
40 NEXT n
```

(Fuente "0602.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

que escribiría lo siguiente en pantalla:

```
1
2
3
4
5
6
7
8
```

Ejercicio propuesto 6.1: *Crea un programa que escriba los números del 10 al 20.*

O podemos escribir la tabla de multiplicar del 5 así:

```
10 REM Tabla de multiplicar del 5
20 FOR factor = 1 TO 10
30 PRINT "5 x "; factor; " vale: ";
40 PRINT factor * 5
50 NEXT factor
```

(Fuente "0603.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Msx Basic 2.1)

El resultado de este programa sería:

```
5 x 1 vale: 5
5 x 2 vale: 10
5 x 3 vale: 15
5 x 4 vale: 20
5 x 5 vale: 25
5 x 6 vale: 30
5 x 7 vale: 35
5 x 8 vale: 40
5 x 9 vale: 45
5 x 10 vale: 50
```

(Nota: en el BASIC de los Commodore 64, no se permite usar como contador la palabra "factor"; si se cambia por "fact" o alguna más breve, funcionará correctamente).

Ejercicio propuesto 6.2: *Crea un programa que escriba la tabla de multiplicar de un número escogido por el usuario.*

No necesariamente tenemos que contar **de uno en uno**. Para ello se emplea la orden **STEP** (paso, en inglés). Por ejemplo, podemos escribir los números pares del 2 al 20 así:

```
10 REM Numeros pares del 1 al 20
20 FOR n = 2 TO 20 STEP 2
30 PRINT n
40 NEXT n
```

(Fuente "0604.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

Ejercicio propuesto 6.3: *Crea un programa que escriba los 10 primeros múltiplos de 3, usando STEP (3, 6, 9, ... hasta llegar a 30).*

También podemos contar "hacia atrás", si usamos un paso negativo:

```
10 REM Descontar desde 10 hasta 0
20 FOR x = 10 TO 0 STEP -1
30 PRINT x
40 NEXT x
```

(Fuente "0605.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

Ejercicio propuesto 6.4: *Crea un programa que escriba tantos asteriscos (en la misma línea) como indique el usuario.*

Podemos **incluir un FOR dentro de otro**. Por ejemplo, podríamos escribir la tabla de multiplicar completa (de todos los números del 1 al 10) así:

```
10 REM Tabla de multiplicar completa
20 FOR tabla = 1 TO 10
30   FOR factor = 1 TO 10
40     PRINT tabla; " x "; factor; " vale: ";
50     PRINT tabla * factor
60   NEXT factor
70   PRINT : REM Linea en blanco entre tablas
80 NEXT tabla
```

(Fuente "0606.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Msx Basic 2.1)

(Nota: nuevamente, en el BASIC de los Commodore 64, no podremos usar la variable "factor"; que deberíamos cambiar "fact" o alguna más breve).

En este programa he ido escribiendo un poco más a la derecha lo que dependía de cada FOR, para intentar que resulte más legible. Esto es lo que se conocer como "**escritura indentada**", y yo lo comenzaré a emplear a partir de ahora.

El resultado de este programa debería ser algo así:

```
1 x 1 vale: 1
1 x 2 vale: 2
1 x 3 vale: 3

[... (muchas mas lineas aqui en medio) ...]

9 x 7 vale: 63
9 x 8 vale: 72
9 x 9 vale: 81
9 x 10 vale: 90

10 x 1 vale: 10
10 x 2 vale: 20
10 x 3 vale: 30
10 x 4 vale: 40
10 x 5 vale: 50
10 x 6 vale: 60
10 x 7 vale: 70
10 x 8 vale: 80
10 x 9 vale: 90
10 x 10 vale: 100
```

Ejercicio propuesto 6.5: *Crea un programa que escriba un rectángulo de asteriscos, con la anchura y altura que indique el usuario. Sería algo como:*

```
Anchura? 5
Altura? 3
*****
*****
*****
```

En la mayoría de las versiones de Basic no es necesario detallar después de la palabra **NEXT** el nombre de la variable a la que se refiere, porque el propio intérprete de Basic será capaz de deducirlo:

```
10 REM Numeros pares del 1 al 20
20 FOR n = 2 TO 20 STEP 2
30 PRINT n
40 NEXT : REM Se sobreentiende que es "n"
```

(Fuente "0607.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

Aun así, en programas grandes, puede ser interesante conservar el nombre de la variable, buscando una mayor legibilidad.

Nota: Los ejemplos de este apartado deberían funcionar en Basic256 sin grandes cambios. Apenas sería necesario recordar quitar los números de línea y que no se pueden escribir varios textos distintos con una misma orden PRINT, de modo que la tabla de multiplicar se escribiría con

```
REM Tabla de multiplicar del 5
FOR factor = 1 TO 10
PRINT "5 x ";
PRINT factor;
PRINT " vale: ";
PRINT factor * 5
NEXT factor
```

(Fuente "0608.bas". Probado en Basic256)

Ejercicio propuesto 6.6: *Crea un programa que te pida tu nombre y un número, y escriba tu nombre tantas veces como indique ese número.*

Ejercicio propuesto 6.7: *Crea un programa que escriba un triángulo decreciente formado por asteriscos, cuya altura escogerá el usuario, así:*

****
***
**
*

Más adelante veremos otras órdenes más sofisticadas para controlar partes del programa que se repitan, como es el caso de WHILE.

7. Comprobación Básica De Condiciones: IF

La forma básica de **comprobar condiciones** es con la construcción "**IF** condición **THEN** orden". Un ejemplo de su uso podría ser:

```
10 PRINT "Introduzca un numero"
20 INPUT x
30 IF x > 10 THEN PRINT "Es mayor que 10"
```

(Fuente "0701.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

Su equivalente en Basic256 sería idéntico, pero sin números de línea:

```
PRINT "Introduzca un numero"
INPUT x
IF x > 10 THEN PRINT "Es mayor que 10"
```

(Fuente "0702.bas". Probado en Basic256)

Ejercicio propuesto 7.1: *Crea una programa que te pida un número y diga si es positivo.*

Alguna versión de BASIC antigua puede ser un poco más restrictiva y permitir sólo la sintaxis "IF condición THEN GOTO linea", que veremos en el siguiente apartado. De momento vamos a suponer que no es el caso, y profundizar más en el uso de IF.

La mayoría de las versiones de BASIC permiten indicar qué hacer si no se cumple la condición, con la orden **ELSE**:

```
10 PRINT "Introduzca un numero"
20 INPUT x
30 IF x > 10 THEN PRINT "Mayor que 10" ELSE PRINT "Menor o igual que 10"
```

(Fuente "0703.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Msx Basic 2.1; Commodore 64 Basic V2 no aceeta "ELSE")

Ejercicio propuesto 7.2: *Crea un programa que te pida un número y diga si es positivo o no lo es, usando IF y ELSE.*

De hecho, normalmente podremos enlazar **varias condiciones** (aunque debemos recordar que en muchas versiones de BASIC antiguas no podremos superar los 255 caracteres por línea). Por ejemplo, podemos comprobar si un número es mayor que 10, menor que 10 o exactamente igual a 10 con:

```
10 PRINT "Introduzca un numero"
20 INPUT x
30 IF x > 10 THEN PRINT "Mayor que 10" ELSE IF x < 10 THEN PRINT "Menor que 10" E
LSE PRINT "Exactamente 10"
```

(Fuente "0704.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Msx Basic 2.1)

Ejercicio propuesto 7.3: *Crea un programa que te pida un número y diga si es negativo, es cero o es positivo, usando dos IF encadenados.*

Nuevamente, en Basic256 sería muy parecido, pero sin números de línea. Esta sería la primera versión, con una única condición:

```
PRINT "Introduzca un numero"
INPUT x
IF x > 10 THEN PRINT "Mayor que 10" ELSE PRINT "Menor o igual que 10"
```

(Fuente "0705.bas". Probado en Basic256)

Pero la segunda versión, con dos condiciones encadenadas, sería válida en QBasic y (con ligeras modificaciones que veremos pronto) FreeBasic, pero no en Basic256:

```
PRINT "Introduzca un numero"
INPUT x
IF x > 10 THEN PRINT "Mayor que 10" ELSE IF x < 10 THEN PRINT "Menor que 10" ELSE
PRINT "Exactamente 10"
```

(Fuente "0706.bas". Probado en QBasic 1.1)

Los BASIC modernos, que trabajan sin números de línea, suelen permitir otra forma ampliada de escribir la orden IF, que permite dar varios pasos si se cumple la condición o si no se cumple, y que termina con **ENDIF**:

```
IF condición THEN
lista_de_ordenes_1
ELSE
lista_de_ordenes_2
ENDIF
```

De modo que, en Basic256 la primera versión del programa anterior se podría escribir también así:

```
PRINT "Introduzca un numero"
INPUT x
IF x > 10 THEN
    PRINT "Mayor de 10"
ELSE
    PRINT "Menor o igual que 10"
ENDIF
```

(Fuente "0707.bas". Probado en Basic256)

Y la segunda podría ser:

```
PRINT "Introduzca un numero"
INPUT x
IF x > 10 THEN
    PRINT "Mayor que 10"
ELSE
    IF x < 10 THEN
        PRINT "Menor que 10"
    ELSE
        PRINT "Exactamente 10"
    ENDIF
ENDIF
```

(Fuente "0708.bas". Probado en Basic256)

Ejercicio propuesto 7.4: *Crea una programa que te pida un número y diga si es negativo, es cero o es positivo, usando dos IF-ENDIF encadenados.*

Hemos empleado el signo ">" para ver si una variable es mayor que un cierto valor y el signo "<&" para ver si es menor. Veamos cómo se indican las otras **comparaciones** posibles:

```
> Mayor que
>= Mayor o igual que
< Menor que
<= Menor o igual que
= Igual a
<> Distinto de
```

Las condiciones se pueden enlazar con **AND** (y), **OR** (ó), **NOT** (no), pero entonces puede ser interesante emplear paréntesis, para que el resultado sea más legible (y para evitar problemas de precedencia de operadores, que no trataremos todavía), por ejemplo:

```
IF ((a>2) AND (b<>3)) OR (NOT (c>3)) THEN PRINT "Es valido"
```

Un programa que comprobase si el usuario teclea 3 o 5 podría ser

```
10 REM Comprobar dos posibles valores
20 PRINT "Introduzca 3 o 5"
30 INPUT x
40 IF x = 3 OR x = 5 THEN PRINT "Correcto!" ELSE PRINT "No!"
```

(Fuente "0709.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Msx Basic 2.1)

Nota: un **error frecuente** es "abreviar demasiado", sin indicar el nombre de la variable en todas las comparaciones, como veremos en el siguiente ejemplo. Algunas versiones de BASIC marcarán esa línea como incorrecta, mientras que otras pueden creer que es correcta, pero dar un resultado incorrecto, porque "OR" puede tener otros significados, que veremos más adelante:

```
10 REM Comprobacion incorrecta dos posibles valores
20 PRINT "Introduzca 3 o 5"
30 INPUT x
40 IF x = 3 OR 5 THEN PRINT "Correcto!" ELSE PRINT "No!": REM Incorrecto!
```

(Fuente "0710.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Msx Basic 2.1)

(¡Este programa, cuya línea 40 está mal escrita, responderá “Correcto!” con otros valores que no son el 3 ni el 5. Más adelante veremos los motivos.)

Ejercicio propuesto 7.5: *Crea una programa que te pida dos número y diga si los dos son positivos o si, por el contrario, al menos uno es negativo o cero.*

Obviamente, se pueden **"mezclar" los "if" y los "for"** en un mismo programa. Por ejemplo, podríamos saludar varias veces a un usuario si su respuesta es correcta:

```
10 PRINT "Dime tu nombre"
20 INPUT nombre$
30 IF nombre$="Nacho" THEN FOR i = 1 TO 5:PRINT"HOLA":NEXT i
```

(Fuente "0711.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

Que en Basic256 sería casi idéntico:

```
PRINT "Dime tu nombre"
INPUT nombre$
IF nombre$="Nacho" THEN FOR i = 1 TO 5:PRINT"HOLA":NEXT i
```

(Fuente "0712.bas". Probado en Basic256)

O bien, desglosando en líneas separadas

```
PRINT "Dime tu nombre"
INPUT nombre$
IF nombre$="Nacho" THEN
    FOR i = 1 TO 5
        PRINT"HOLA"
    NEXT i
END IF
```

(Fuente "0713.bas". Probado en Basic256)

Y también podemos comprobar condiciones dentro de un "for". Por ejemplo, podemos tratar de encontrar los múltiplos de 7 que hay entre 1 y 100, si recorremos los números del 1 al 100, mirando para cada uno de ellos si la división entre 7 es exacta:

```
10 REM Multiplos de 7
20 FOR i = 1 TO 100
30   IF i/7 = INT(i/7) THEN PRINT i
40 NEXT i
```

(Fuente "0714.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

Ejercicio propuesto 7.6: *Crea un programa que te pida un número (entero) y muestre sus divisores.*

Ejercicio propuesto 7.7: *Crea un programa que te pida un número (entero) y diga si es primo.*

Ejercicio propuesto 7.8: *Crea un programa que muestre los números primos entre 1 y 100.*

Ejercicio propuesto 7.9: *Crea un programa que muestre los números primos entre 1 y el número que escoja el usuario.*

Ejercicio propuesto 7.10: *Crea un programa que muestre los números entre 1 y 100 que sean múltiplos de 2 y de 3 pero no de 12.*

8. Saltos: GOTO Y GOSUB

En ocasiones nos puede interesar saltar a una parte distinta del programa. Esto será especialmente útil después de comprobar una cierta condición, por ejemplo: "si las ganancias han sido superiores a 1000, saltar a la parte de programa que decide cómo desglosar los beneficios".

Para saltar usaremos la orden **GOTO** (del inglés "go to", ir a; de hecho, alguna versión de BASIC permite o incluso exige que GO TO se escriba como dos palabras separadas). Después de esta orden detallaremos el número de línea a la que deseamos saltar. Un ejemplo puede ser el siguiente programa, que escribe indefinidamente la palabra "Hola", porque la línea 40 hace que el programa vuelva a la línea 30 una y otra vez:

```
10 REM Un programa sin salida
20 REM Cuidado - se quedara "colgado"
30 PRINT "Hola"
40 GOTO 30
```

(Fuente "0801.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

Es muy fácil que este programa no se pueda interrumpir y deje el ordenador "colgado". En algún ordenador (y alguna versión de BASIC), se podría interrumpir pulsando dos veces la tecla Esc, o las teclas Ctrl+C, o alguna otra combinación similar de teclas. En otros puede no haber más remedio que apagar el equipo.

Basic256 (y alguna otra versión moderna de BASIC) no permite números de líneas, pero aun así permite usar GOTO, si indicamos una etiqueta (una palabra terminada con el símbolo de "dos puntos") a la que saltar:

```
REM Un programa SIN salida
REM Cuidado - se quedara "colgado"
repetir: PRINT "Hola"
GOTO repetir
```

(Fuente "0802.bas". Probado en Basic256)

En este caso, el programa no terminará hasta que no pulsemos el botón "STOP" de nuestro entorno de desarrollo.

Ejercicio propuesto 8.1: *Crea un programa que escriba tu nombre una cantidad indeterminada de veces (hasta que el usuario lo interrumpa).*

Es importante recordar que GOTO es un salto incondicional, de modo que podemos "saltarnos" un trozo de un programa sin darnos cuenta si no llevamos cuidado, como en el siguiente ejemplo:

```
10 REM Este programa no escribe "Hola"
20 GOTO 40
30 PRINT "Hola"
40 PRINT "Adios"
```

(Fuente "0803.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

Si nos equivocamos y saltamos a una línea, que no existe, el intérprete de BASIC daría un mensaje de error parecido a "Undefined line 50" o "Line does not exist in 10"

```
10 PRINT "Hola"
20 GOTO 50
```

(Fuente "0804.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

Ejercicio propuesto 8.2: *Crea un programa que escriba una cantidad indeterminada de asteriscos, sin avanzar de línea (hasta que el usuario lo interrumpa).*

Es muy habitual usar GOTO en conjunción con IF, para hacer que un bloque de un programa se repita mientras se cumpla una cierta condición, por ejemplo:

```
10 PRINT "Introduzca el numero 5"
20 INPUT x
30 IF x <> 5 THEN GOTO 10
40 PRINT "Correcto!"
```

(Fuente "0805.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

(Dentro de poco veremos alternativas más elegantes a este uso de GOTO)

Además, alguna versión de BASIC antigua puede permitir sólo la sintaxis "IF condición THEN GOTO linea" o cualquiera de dos versiones abreviadas de la misma: "IF condición THEN linea" o "IF condición GOTO linea", y no permitir ninguna otra orden en una condición IF. En este caso, el programa 0701, que decía si un número es mayor que 10, sería más incómodo y menos legible:

```
10 PRINT "Introduzca un numero"
20 INPUT x
30 IF x > 10 THEN GOTO 50
40 END
50 PRINT "Es mayor que 10"
```

(Fuente "0806.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

Ejercicio propuesto 8.3: *Crea un programa que te pida un número y diga si es negativo, es cero o es positivo, usando GOTO en vez de ELSE*

La orden GOTO tiene una gran carencia: el programa sigue avanzando después del salto, como si no hubiera ocurrido ningún salto. Pero a veces no es eso lo que más nos interesa, sino que nos encontraremos con tareas repetitivas, y querremos que nuestro programa salte a una zona en la que le hayamos indicado cómo dar esos pasos repetitivos, pero después vuelva y siga trabajando normalmente. En estos casos en que **queremos que salte y vuelva**, emplearemos la orden **GOSUB**, e indicaremos el punto de retorno con **RETURN**, así:

```
10 ' Ejemplo de GOSUB y RETURN
20 ' Vamos a saludar
30 GOSUB 100
40 PRINT "Juan"
50 GOSUB 100: ' volvemos a saludar
60 PRINT "Alberto"
70 END
100 PRINT "Hola"
110 PRINT "Como estas?"
120 RETURN
```

(Fuente "0807.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Msx Basic 2.1, Commodore 64 Basic V2 -si se cambian los apóstrofes por REM-)

Las líneas 100 a 120 forman la "subrutina" (de ahí viene la palabra GOSUB: de "go subroutine", ir a una subrutina) a la que saltamos cada vez que nos interesa, y de la que volvemos para que el programa siga por donde estaba. En este caso hemos indicado dónde está el final del programa con la orden END, para evitar que después de la línea 60 el programa siga avanzando y pase a la línea 100 (la subrutina) sin que nosotros se lo pidamos.

El programa anterior sería equivalente a este otro:

```
30 PRINT "Hola"
35 PRINT "Como estas?"
40 PRINT "Juan"
50 PRINT "Hola"
55 PRINT "Como estas?"
60 PRINT "Alberto"
```

(Fuente "0808.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

He eliminado las líneas con REM, que sólo son aclaraciones para nosotros; cada GOSUB es equivalente a que se dieran todos los pasos que hay entre el punto de comienzo del salto (en este caso, la línea 100) y la línea de RETURN.

El equivalente en Basic256 del primer programa con GOSUB sería

```
REM Ejemplo de GOSUB y RETURN
REM Vamos a saludar

GOSUB saludo
PRINT "Juan"

GOSUB saludo
PRINT "Alberto"

END

saludo:
PRINT "Hola"
PRINT "Como estas?"
RETURN
```

(Fuente "0809.bas". Probado en Basic256)

Ejercicio propuesto 8.4: *Crea una subrutina que escriba 10 asteriscos en la misma línea. Úsala para pedir al usuario 3 nombres y mostrarlos "subrayados".*

Existe otro uso adicional de GOSUB en los BASIC antiguos: si "hacemos demasiadas cosas" en la misma línea de un IF, no sólo el programa acabará resultando más difícil de leer, sino que, además, en muchos BASIC clásicos había un límite de tamaño relativamente pequeño para cada línea de programa, típicamente cerca de 256 letras. Por eso, si tenemos que hacer muchas cosas que dependan de la condición, será mejor moverlas a una subrutina:

```
10 PRINT "Dime tu nombre"
20 INPUT nombre$
30 IF nombre$="Nacho" THEN GOSUB 100
40 END
100 REM Saludar 5 veces
110 FOR i = 1 TO 5
120   PRINT"HOLA"
130 NEXT i
140 RETURN
```

(Fuente "0810.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

Notas: Casi todas las versiones de BASIC permiten la orden GOTO; pero algunas antiguas no permiten GOSUB. En algunas versiones de Basic se podía (o incluso se debía) escribir estas órdenes como dos palabras separadas: GO TO y GO SUB.

Las órdenes GOTO y GOSUB se usan cada vez menos, porque en versiones modernas de BASIC tendremos alternativas más elegantes para estructuras repetitivas, como la orden WHILE, que veremos pronto. También tendremos formas más eficientes de crear subrutinas, que conoceremos poco después.

9. Guardar Y Recuperar Programas. Borrar El Programa En Memoria: SAVE, LOAD, NEW

Cuando apagamos el ordenador, **se borra toda la información** que hay en la memoria convencional. Por eso, hay que **guardar los programas** en el disco duro o en un diskette (o incluso en cinta magnética, si el ordenador es muy antiguo).

La forma de grabar un programa en un BASIC clásico es teclear la orden **SAVE** y entre comillas el nombre que queramos dar al programa (que normalmente estará limitado a unas 8 letras y no podrá contener espacios), así:

```
SAVE "PRUEBA1"
```

(Fuente "0901.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Msx Basic 2.1)

En los BASIC más modernos, que tengan un entorno integrado para escribir y corregir los programas, nuestro programa se grabará desde el menú, normalmente desde el **submenú "Archivo"** o "File".

Una vez que volvamos a encender nuestro ordenador y queramos **recuperar** el programa que habíamos guardado, usaremos la orden **LOAD**, indicando nuevamente entre comillas el nombre del programa a recuperar:

```
LOAD "PRUEBA1"
```

(Fuente "0902.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Msx Basic 2.1)

En los BASIC más modernos, el programa se recuperará desde el menú, normalmente desde el submenú "Archivo" o "File".

En los ordenadores clásicos más antiguos, que no tenían unidad de diskette, sino de **cinta magnética**, se solía permitir teclear LOAD sin indicar nombre de fichero, de modo que se recuperaría el primero que se encontrase en la cinta:

```
LOAD ""
```

(Fuente "0903.bas". Probado en Locomotive Basic de Amstrad CPC)

Si queremos **eliminar el programa que tenemos en memoria**, para empezar a teclear uno nuevo, hay al menos dos formas de conseguirlo: la "mala" es salir del intérprete de BASIC (o apagar el ordenador) y volver a entrar; la "buena" es emplear la orden **NEW**:

```
NEW
```

(Fuente "0904.bas". Probado en Locomotive Basic de Amstrad CPC, Msx Basic 2.1)

Según la versión de BASIC, esta orden puede borrar el programa y las variables que hayamos definido, o solo el programa, pero no las variables. Esto puede suponer que si probamos un programa como éste:

```
10 PRINT "Introduce un numero"
20 INPUT x
30 PRINT x
```

(Fuente "0905.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Msx Basic 2.1)

si introducimos el valor 20, borramos el programa con la orden NEW, y después volvemos a escribir PRINT X, hay muchas versiones de BASIC antiguas que todavía escribirán 20. Es el caso de los Amstrad CPC y Commodore 64; por el contrario, en MSX BASIC 2.1 sí se borran las variables tras el NEW.

Si queremos **borrar también los valores de las variables**, se puede emplear la orden **CLEAR**(nuevamente, no existe en todas las versiones de BASIC).

Nuevamente, en los BASIC más modernos, se crea un programa nuevo desde el menú, normalmente desde el submenú "Archivo" o "File", y todas las variables quedarán vacías.

Ejercicio propuesto 9.1: *Crea un programa que muestra la suma de tres números introducidos por el usuario. Guárdalo con el nombre "SUMA3.BAS". Después borra la memoria con NEW (si usas un BASIC clásico) o sal del entorno y vuelve a entrar (si usas uno moderno) y vuélvelo a cargar con LOAD (o desde el menú).*

10. Funciones Predefinidas En BASIC

En el lenguaje BASIC contamos con una serie de funciones predefinidas que nos pueden resultar muy útiles. Unas son funciones matemáticas, como el valor absoluto o el coseno de un ángulo; otras nos permitirán escoger parte de una cadena de texto (por ejemplo las letras de más a la izquierda), etc. Veremos las más habituales.

Las funciones **matemáticas** que encontraremos casi en cualquier intérprete de BASIC son:

```
ABS - Devuelve el valor absoluto de un número.
ATN - Devuelve el arco tangente.
COS - Devuelve el coseno de un ángulo (medido en radianes).
SIN - Devuelve el seno de un ángulo (medido en radianes).
LOG - Devuelve el logaritmo natural o neperiano (base e).
EXP - Devuelve la exponencial (e elevado al número).
RND - Devuelve un número aleatorio entre 0 y 1 (se deberá
usar junto con RANDOMIZE).
SQR - Devuelve la raíz cuadrada de un número.
INT - Devuelve el mayor número entero que es menor o igual
que el número dado.
```

En cuanto a los **números aleatorios** (generados al azar), la orden **RANDOMIZE** es la que hace que se comiencen a generar a partir de una cierta "semilla". Se usa "RANDOMIZE numero". El problema es que si escribimos, por ejemplo, RANDOMIZE 2, siempre obtendremos la misma secuencia de números "aleatorios". Si queremos números realmente "al azar", la mejor solución suele ser basarse en el reloj interno del sistema (si el ordenador lo tiene), dado que será casi imposible que el ordenador se ponga siempre en marcha a la misma hora (con una precisión de centésimas de segundo). Eso, en QBasic, se consigue escribiendo RANDOMIZE TIMER.

Un ejemplo que pruebe estas funciones sería:

```
10 REM Funciones matematicas
20 PRINT "Valor absoluto de -5: "; ABS(-5)
30 PRINT "Coseno de 0,5 radianes: "; COS(0.5)
40 PRINT "Seno de 0,5 radianes: "; SIN(0.5)
50 PRINT "Arco cuya tangente vale 1: "; ATN(1)
60 PRINT "Logaritmo natural de 10: "; LOG(10)
70 PRINT "El número e vale: "; EXP(1)
80 PRINT "La raiz de 125 es: "; SQR(125)
90 PRINT "La parte entera de 4,7 es: "; INT(4.7)
100 RANDOMIZE TIMER
110 PRINT "Un número al azar: "; RND
```

(Fuente "1001.bas". Probado en GwBasic 2.02, QBasic 1.1; la gran mayoría de versiones de BASIC aceptarán las líneas hasta la 90 inclusive, pero pueden variar en la forma de generar números al azar: Msx Basic 2.1 y Commodore 64 Basic V2 reconocen las líneas 10 a 90)

Y su resultado (con QBasic) es:

```
Valor absoluto de -5: 5
Coseno de 0,5 radianes: .8775826
Seno de 0,5 radianes: .4794255
Arco cuya tangente vale 1: .7853982
Logaritmo natural de 10: 2.302585
El número e vale: 2.718282
La raiz de 125 es: 11.18034
La parte entera de 4,7 es: 4
Un número al azar: .5464441
```

Su equivalente en Basic256 sería muy parecido, con las diferencias de que no se puede escribir varias expresiones en un mismo PRINT, que no existe ATN y que para obtener un número al azar ser usa RAND en vez de RND, y no es necesario hacer un RANDOMIZE TIMER para inicializar la semilla:

```
REM Funciones matematicas
PRINT "Valor absoluto de -5: ";
PRINT ABS(-5)
PRINT "Coseno de 0,5 radianes: ";
PRINT COS(0.5)
PRINT "Seno de 0,5 radianes: ";
PRINT SIN(0.5)
PRINT "Logaritmo natural de 10: ";
PRINT LOG(10)
PRINT "El número e vale: ";
PRINT EXP(1)
PRINT "La raiz de 125 es: ";
PRINT SQR(125)
PRINT "La parte entera de 4,7 es: ";
PRINT INT(4.7)
PRINT "Un número al azar: ";
PRINT RAND
```

(Fuente "1002.bas". Probado en Basic256)

Ejercicios propuestos:

Ejercicio propuesto 10.1: *Crea un programa que calcule la raíz cuadrada del número que introduzca el usuario.*

Ejercicio propuesto 10.2: *Crea un programa que te muestre el coseno de PI radianes (recuerda que el valor de PI es 3.141593).*

Ejercicio propuesto 10.3: *Haz un programa que calcule el seno de 45 grados (recuerda la equivalencia: 180 grados = PI radianes)*

Ejercicio propuesto 10.4: *Escribe un programa que calcule las soluciones de una ecuación de segundo grado de la forma* $Ax^2+Bx+C=0$

Ejercicio propuesto 10.5: *Crea un programa que genere un número al azar entre 50 y 100 (ambos incluidos)*

Ejercicio propuesto 10.6: *Crea un programa que dé al usuario 3 oportunidades que para adivinar un número (generado al azar) del 1 al 10*

Por otra parte, las órdenes para manipular **cadenas de texto** que permitirán casi todas las versiones de BASIC son:

- LEN - Longitud de la cadena de texto: PRINT LEN("Nacho") escribiría 5.
- VAL - Valor numérico de la cadena, para poder hacer operaciones con ella.
- RIGHT$ - Toma varias letras de la derecha de la cadena: PRINT RIGHT$("Nacho", 3) escribiría "cho".
- LEFT$ - Toma varias letras de la izquierda de la cadena: PRINT LEFT$("Nacho", 2) escribiría "Na".
- MID$ - Toma varias letras intermedias de la cadena: PRINT MID$("Nacho", 2,3) escribiría "ach"
- CHR$ - Letra o símbolo con un cierto código ASCII: PRINT CHR$(65) escribiría "A"
- ASC - Dice el código ASCII de una letra o símbolo: PRINT ASC("A") escribiría 65.
- INSTR - Indica en qué posición de una cadena se encuentra cierto texto: PRINT INSTR("Nacho", "ach") diría "2". Si no existe, el resultado será 0. Se puede indicar un dato adicional, que es a partir de qué posición queremos buscar (este dado, si se incluye, debe ser el primero): PRINT INSTR(4, "Nacho", "ach") diría "0".

(Nota: el código ASCII es un estándar que permite guardar la información dentro del ordenador, representando internamente cada letra con un cierto número).

Un ejemplo sería:

```
10 REM Funciones de cadena
20 texto1$ = "Texto de prueba"
30 texto2$ = "23"
40 PRINT "La longitud del texto 1 es: "; LEN(texto1$)
50 PRINT "Las primeras 5 letras son: "; LEFT$(texto1$, 5)
60 PRINT "Las 6 ultimas son: "; RIGHT$(texto1$, 6)
70 PRINT "De la 3 a la 7 son: "; MID$(texto1$, 3, 5)
80 PRINT "El doble del valor numérico del texto 2 es: "; VAL(texto2$)*2
90 PRINT "El codigo ASCII de la A es: "; ASC("A")
100 PRINT "La letra 99 es: "; CHR$(99)
110 PRINT "La posición de A dentro de HOLA es: "; INSTR("HOLA","A")
```

(Fuente "1003.bas". Probado en GwBasic 2.02, QBasic 1.1; en Commodore 64 Basic V2 son válidas las líneas 10 a 100 si se renombran texto1$ y texto2$ a t1$ y t2$, en Msx Basic 2.1 son válidas todas ellas si se renombran así)

Cuyo resultado será:

```
La longitud del texto 1 es: 15
Las primeras 5 letras son: Texto
Las 6 ultimas son: prueba
De la 3 a la 7 son: xto d
El doble del valor numérico del texto 2 es: 46
El codigo ASCII de la A es: 65
La letra 99 es: c
```

En Basic256 hay algunas diferencias:

- Las dos peculiaridades que ya conocíamos: no se puede usar números de línea y no se puede encadenar varias expresiones separadas por "punto y coma" en una misma orden PRINT
- LEN se llama LENGTH
- LEFT$, RIGHT$, MID$ y CHR$ no usan símbolo de "dólar", y se convierten en LEFT, RIGHT, MID y CHR
- No existe VAL, pero en su lugar se puede usar INT

Por tanto, el equivalente de este programa sería:

```
REM Funciones de cadena
texto1$ = "Texto de prueba"
texto2$ = "23"
PRINT "La longitud del texto 1 es: ";
PRINT LENGTH(texto1$)
PRINT "Las primeras 5 letras son: ";
PRINT LEFT(texto1$, 5)
PRINT "Las 6 ultimas son: ";
PRINT RIGHT(texto1$, 6)
PRINT "De la 3 a la 7 son: ";
PRINT MID(texto1$, 3, 5)
PRINT "El doble del valor numérico del texto 2 es: ";
PRINT INT(texto2$)*2
PRINT "El codigo ASCII de la A es: ";
PRINT ASC("A")
PRINT "La letra 99 es: ";
PRINT CHR(99)
PRINT "La posición de A dentro de HOLA es: ";
PRINT INSTR("HOLA","A")
```

(Fuente "1004.bas". Probado en Basic256)

Ejercicio propuesto 10.7: *Crea un programa que pida una frase al usuario y la escriba sin la primera letra ni la última, usando MID$*

Ejercicio propuesto 10.8: *Crea un programa que pida al usuario su nombre y lo escriba con espacios entre las letras (por ejemplo, a partir de "NACHO" escribiría "N A C H O")*

Ejercicio propuesto 10.9: *Haz un programa que pida al usuario su nombre y lo escriba al revés (por ejemplo, a partir de "NACHO" escribiría "OHCAN")*

Ejercicio propuesto 10.10: *Crea un programa que pida una frase al usuario y la escriba sin espacios (por ejemplo. "COMO ESTAS?" se convertiría en "COMOESTAS?")*

Ejercicio propuesto 10.11: *Crea un programa escriba un triángulo a partir del nombre del usuario. Por ejemplo, a partir de "NACHO" escribiría:*
N
NA
NAC

NACH
NACHO

Ejercicio propuesto 10.12: *Crea un programa escriba las letras que equivalen a los códigos ASCII desde el 32 hasta el 127. Para cada una de estas letras se escribirá el número, un símbolo de igual, el carácter equivalente y finalmente un espacio en blanco, que actuará de separador.*

11. Otros Bucles: While, Loop

Hemos visto cómo emplear la orden FOR para hacer que una parte de un programa se repita incondicionalmente, una cierta cantidad de veces. También hemos comentado cómo comprobar condiciones con la orden IF.

Pero en la práctica es muy frecuente que nos encontremos una "mezcla" de ambas cosas: una serie de operaciones que se deben **repetir mientras se cumpla una cierta condición**.

En los BASIC más antiguos, estos se puede conseguir si utilizamos juntas la orden **IF** y la orden **GOTO**. Por ejemplo, un programa que pidiese una contraseña, para restringir el acceso a personas no deseadas, podría ser así:

```
10 PRINT "ESCRIBE TU CLAVE DE ACCESO"
20 INPUT clave$
30 IF clave$ <> "xjk5" THEN GOTO 10
```

(Fuente "1101.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

Nota: muchas versiones de BASIC permitirían escribir también la línea 40 de alguna -o cualquiera- de estas formas:

```
30 IF clave$ <> "xjk5" THEN 10
30 IF clave$ <> "xjk5" GOTO 10
```

Este programa es corto y todavía resulta fácil de leer, pero en un programa mínimamente grande, si lo llenamos de saltos con la orden GOTO, termina siendo totalmente ilegible y muy difícil de seguir. Por eso, algunos BASIC más modernos incluían **otras formas más elegantes** de conseguir lo mismo.

La primera (que pocas versiones de BASIC de los años 80 permitían) es la orden **WHILE**, cuyo formato es

```
WHILE condición
  órdenes
WEND
```

De modo que podríamos reescribir nuestro programa de una forma mucho más legible:

```
10 WHILE clave$ <> "xjk5"
20   PRINT "ESCRIBE TU CLAVE DE ACCESO"
30   INPUT clave$
40 WEND
```

(Fuente "1102.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC)

Lo de escribir las líneas 20 y 30 más a la derecha es sólo buscando que el programa resulte más fácil de leer, como ya vimos en el apartado 6. Los intérpretes de BASIC "ignorarán" estos espacios en blanco (aunque puede que alguno "los borre"), de modo que "no estorban", pero a cambio conseguiremos que se vea mejor dónde empieza y dónde terminar la parte que se repite. (Este tipo de "sangrados" en un programa es lo que se conoce como **escritura indentada**).

En este programa hay una cosa peligrosa: estamos comprobando el valor de la variable clave$ antes de que se haya usado. En la mayoría de las versiones de BASIC, la primera vez que usamos una variable, se le da automáticamente el valor 0 (si es numérica) o el de una cadena de texto vacía (si es texto), de modo que no tendríamos problemas. Aun así, no es mala costumbre "dejarlo claro" en el propio programa, para evitar que quede "basura" de otras veces que hubiéramos utilizado nuestro programa:

```
10 clave$ = "" : REM Aseguramos que la cadena este vacia
20 WHILE clave$ <> "xjk5"
30 PRINT "ESCRIBE TU CLAVE DE ACCESO"
40 INPUT clave$
50 WEND
```

(Fuente "1103.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC)

En Basic256 no existe "WEND" para cerrar la parte repetitiva, sino que se usa "END WHILE", de modo que el programa quedaría:

```
REM Primero aseguramos que la cadena esté vacía
clave$ = ""
REM Y después comienza la parte repetitiva
WHILE clave$ <> "xjk5"
    PRINT "ESCRIBE TU CLAVE DE ACCESO"
    INPUT clave$
END WHILE
 nbsp;
```

(Fuente "1104.bas". Probado en Basic256)

Algunas versiones más modernas de BASIC (como QBasic) permiten **otra opción más elegante** y con más posibilidades: la construcción **DO..LOOP**, que se puede emplear de varias formas:

```
DO WHILE condición
  órdenes
LOOP
```

(se repite MIENTRAS se cumpla la condición, pero esa condición se comprueba antes que nada) o bien:

```
DO
  órdenes
LOOP WHILE condición
```

(se repite MIENTRAS se cumpla la condición, pero esa condición se comprueba tras cada "vuelta") o bien:

```
DO UNTIL condición
  órdenes
LOOP
```

(se repite HASTA QUE se cumpla la condición, pero esa condición se comprueba lo primero de todo) o también:

```
DO
  órdenes
LOOP UNTIL condición
```

(se repite HASTA QUE se cumpla la condición, pero esa condición se comprueba en último lugar)

De modo que la forma más elegante de escribir el **programa anterior** con QBasic y otros BASIC modernos sería:

```
DO
  PRINT "ESCRIBE TU CLAVE DE ACCESO"
  INPUT clave$
LOOP UNTIL clave$ = "xjk5"
```

(Fuente "1105.bas". Probado en QBasic 1.1)

que además coincide con la forma "natural" en que nosotros expresaríamos lo que queremos: algo como "pedir la clave de acceso, hasta que la introducida sea la correcta".

En Basic256 también existe UNTIL, y también comienza con DO, pero no existe la palabra LOOP, por lo que el programa anterior sería:

```
DO
  PRINT "ESCRIBE TU CLAVE DE ACCESO"
  INPUT clave$
UNTIL clave$ = "xjk5"
```

(Fuente "1106.bas". Probado en Basic256)

Ejercicio propuesto 11.1: *Crea un programa que dé al usuario 10 oportunidades que para adivinar un número (generado al azar) del 1 al 1000*

Ejercicio propuesto 11.2: *Haz un programa que pida al usuario su nombre y su contraseña. No le debe dejar continuar hasta que el nombre sea "yo" y la contraseña sea "1234"*

Ejercicio propuesto 11.3: *Crea un programa que pida números al usuario y, para cada número n, escriba el valor de n*10. Deberá repetirse hasta que se introduzca el número 0.*

Ejercicio propuesto 11.4: *Crea un programa que pida números al usuario y, para cada número n, escriba la lista de sus divisores. Deberá repetirse hasta que se introduzca el número 0.*

12. Escribir Con Formato: PRINT USING

Hemos visto que podemos escribir un texto en pantalla si lo indicamos entre comillas en una orden PRINT:

```
PRINT "Bienvenido"
```

y mostrar el valor de una variable si no usamos comillas:

```
PRINT nombre$
```

También sabemos que podemos escribir varias cosas en la misma línea, si incluimos un punto y coma entre dos PRINT (y entonces aparecerán juntas) o si usamos una coma (y entonces aparecen separadas por un cierto número de espacios).

```
10 PRINT "Bienvenido " ; nombre$ , "Pulsa una tecla para seguir"
```

(Fuente "1201.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC)

Pero también podemos indicar que el cursor salte hasta una cierta posición de la línea, usando **TAB**, o decir cuántos espacios queremos que separen una frase de la otra, usando **SPC**, así:

```
10 nombre$="tu"
20 PRINT "Bienvenido" ; TAB(30); nombre$ ;
30 PRINT SPC(10) ; "Pulsa una tecla para seguir"
```

(Fuente "1202.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC, Commodore 64 Basic V2, Msx Basic 2.1)

En la mayoría de versiones de BASIC , estas dos órdenes escribirían la palabra Bienvenido, luego saltarían hasta la columna 30, escribirían el valor de la variable Nombre$, dejarían 10 espacios en blanco y mostrarían el texto final.

Nota: Hay alguna versión de BASIC en la que no existe SPC, y alguna en la que TAB(20) no salta hasta la columna 20, sino que deja 20 espacios en blanco a partir de la posición actual.

Hay otra posibilidad de PRINT que no hemos visto, y que puede ser muy interesante para cuando queramos dar formato a una gran cantidad de números. Se trata de PRINT USING, que nos permite indicar la cantidad de cifras total, la de decimales, si queremos punto en los miles, etc- Vamos a verlo directamente con un ejemplo

```
a = 100
b = 1234.56

PRINT USING "#####.###"; a; b
PRINT USING "#####,.###"; a; b
PRINT USING "#####,"; a; b
PRINT USING "##"; a; b
PRINT USING "+#####.#"; a; b
PRINT USING "**#######.#"; a; b
```

(Fuente "1203.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC)

Da como salida

```
100.000 1234.560
100.000 1,234.560
100 1,235
%100%1235
+100.0 +1234.6
******100.0*****1234.6
```

Debería ser fácil de entender, pero vamos a resumir:

- Cada # indica la posición de una cifra numérica. Por ejemplo, #### dejaría 4 espacios para números.
- Con un punto (.) se indica cuantas posiciones decimales queremos. Por ejemplo, ###.## deja tres espacios antes de la coma y dos posiciones decimales (eso sí, la "coma decimal" aparece como un punto; si estamos en España o en algún otro país que use la coma para los decimales, el resultado no será del todo correcto.
- Una coma (,) a la izquierda del punto decimal indica que también queremos separar los miles (nuevamente, el resultado no será lo que desearíamos en España, porque se separarían con comas, y nosotros usamos puntos).
- Si no hay suficiente hueco para todas las cifras del número, se escribe el número entero, pero se precede por el símbolo % para avisarnos de que no está todo lo colocado que debería, como en el caso de "##"; 100
- Si comenzamos con "+", obligamos a que se muestre el signo, tanto si es positivo (que no sería estrictamente necesario) como si es negativo.
- Si empezamos por **, los espacios en blanco que queden a la izquierda se rellenan con asteriscos.
- (Hay más posibilidades, pero nos vamos a quedar en las más habituales, que son éstas).

Nota: No todas las versiones de BASIC reconocen PRINT USING. No estará en las versiones más antiguas (C64, MSX), pero tampoco existe en Basic256, por ejemplo.

Ejercicio propuesto 12.1: *Pide al usuario 5 datos del 1 al 1000 (quizá con cifras decimales) y muéstralos perfectamente tabulados en pantalla, con dos cifras decimales, de forma que todas las "comas" queden alineadas en una misma vertical.*

13. Más Control De La Pantalla En Modo Texto

En la gran mayoría de las versiones de lenguaje BASIC, podremos borrar la pantalla y escribir texto en distintas posiciones de la pantalla y en distintos colores. Vamos a ver las formas más habituales para conseguirlo...

La orden para borrar la pantalla es **CLS**, y se trata de una orden que casi todas las versiones de BASIC incorporan:

```
CLS
```

(Nota: esta orden no existe en el BASIC de los Commodore 64, pero se puede sustituir por "PRINT CHR$(147)").

La orden para escribir en una cierta **posición de la pantalla** (en "modo texto") no es estándar: cada versión de BASIC tiene la suya propia. Por ejemplo en QBasic se emplea la orden **LOCATE** fila, columna, así:

```
10 LOCATE 10, 20
20 PRINT "hola"
```

(Fuente "1301.bas". Probado en GwBasic 2.02, QBasic 1.1, Locomotive Basic de Amstrad CPC, Msx Basic 2.1)

En el BASIC de los Amstrad CPC y en el de los MSX, el programa anterior es perfectamente válido, pero el primer parámetro indicar la columna y el segundo la fila, al contrario que en GwBasic y QBasic.

Otros intérpretes de Basic pueden emplear otras órdenes, como PRINT AT, PRINT CSR, PRINT @, etc., o puede que escriban la columna antes de la fila.
En el caso de QBasic, además tenemos dos funciones que nos dicen **en qué fila** de la pantalla **está el cursor** (**CSRLIN**) y **en que columna** (**POS**(0)), así:

```
1 REM Pantalla de texto con GwBasic y QBasic
5 CLS
10 LOCATE 10, 30
20 PRINT "hola";
30 fila = CSRLIN
40 columna = POS(0)
50 LOCATE 12, 1
60 PRINT "El cursor estaba en "; fila; ","; columna
```

(Fuente "1302.bas". Probado en GwBasic 2.02, QBasic 1.1, Msx Basic 2.1)

```
Archivo  Edición  Ver  Búsqueda  Ejecutar  Depurar  Opciones  Ayuda
                        LOCATE.BAS
1 ' Pantalla de texto con QBasic_
5 CLS
10 LOCATE 10, 30
20 PRINT "hola";
30 fila = CSRLIN
40 columna = POS(0)
50 LOCATE 12, 1
60 PRINT "El cursor estaba en "; fila; ","; columna
                        Inmediata
Mayús+F1=Ayuda  F6=Ventana  F2=Subs  F5=Ejecutar  F8=Paso      00001:033
```

El resultado de este programa, en QBasic, sería:

```
                          hola
El cursor estaba en  10 , 34

Presione cualquier tecla y continue
```

(Nota: Entre el paréntesis que sigue a POS se podría incluir cualquier expresión, que se va a ignorar, pero por simplicidad era frecuente escribir un cero, como en este ejemplo).

La orden para cambiar el **color de texto** en QBasic es **COLOR pluma, fondo**. A partir de esta orden, se comenzará a escribir en los colores que hayamos indicado, y seguirá así hasta que usemos otra orden COLOR distinta. Para escribir la palabra Hola en letras amarillas y con fondo azul, usaríamos:

```
10 COLOR 14,1
20 PRINT "hola"
```

(Fuente "1303.bas". Probado en GwBasic 2.02, QBasic 1.1, Msx Basic 2.1)

Los colores de texto ("pluma") en QBasic y GwBasic trabajando con un ordenador PC serían los siguientes:

```
0 Negro
1 Azul
2 Verde
3 Azul-verdoso
4 Rojo
5 Magenta
6 Marrón
7 Blanco (gris claro)

8 Gris (oscuro)
9 Azul claro
10 Verde claro
11 Azul-verdoso claro
12 Rojo claro
13 Magenta claro
14 Amarillo
15 Blanco de alta intensidad
```

De modo que podríamos probar estos colores, si los recorremos con un "for":

```
1 REM Colores con GwBasic y QBasic
10 CLS
20 FOR i = 1 TO 15
30   COLOR i
40   LOCATE i, 10
50   PRINT "hola";
60 NEXT
```

(Fuente "1304.bas". Probado en GwBasic 2.02, QBasic 1.1)

En "condiciones normales" (se puede evitar, pero no entraremos en tanto detalle), en un ordenador PC con pantalla a color, los colores 0 a 7 son los que se pueden utilizar como fondo. Si escribimos un color de fondo con un valor entre 8 y 15, se usará el correspondiente entre 0 y 7. Por ejemplo:

```
10 COLOR 9,14
20 PRINT "hola"
```

(Fuente "1305.bas". Probado en GwBasic 2.02, QBasic 1.1)

Será equivalente a escribir COLOR 9, 6, es decir, que el color de primer plano sí que será azul claro, pero el fondo no será amarillo, sino marrón.

En cada **versión de BASIC** cambiarán los detalles concretos de cómo cambiar posiciones y colores en pantalla. Por ejemplo, los ordenadores Amstrad CPC permiten 2 colores en el modo de pantalla de 80 columnas ("mode 2"), 4 colores en el modo de 40 columnas ("mode 1") o 16 colores en el modo de 20 columnas ("mode 0"). Permiten colocarnos en cualquier posición de pantalla con **LOCATE** (pero se indica en primer lugar la columna y después la fila, al contrario que en QBasic), cambiar el color de escritura con

PEN

y el de fondo con **PAPER**, así como la asignación de colores (de entra una paleta de 27 colores posibles con **INK**):

```
1 REM Colores con Amstrad CPC
10 MODE 0
20 FOR i = 1 TO 15
30   PEN i
40   LOCATE 10, i
50   PRINT "hola";
60 NEXT
```

(Fuente "1306.bas". Probado en Locomotive Basic de Amstrad CPC)

Algunos **BASIC modernos** se centran en el modo gráfico y no tienen órdenes avanzadas para trabajar en modo texto. Es el caso de Basic256, por ejemplo, que incorpora CLS y PRINT, pero no permite escribir en distintas coordenadas de la pantalla de texto ni con distintos colores.

Ejercicio propuesto 13.1: *Escribe el texto "Hola" en el centro de la pantalla, en color amarillo.*

Ejercicio propuesto 13.2: *Borra la pantalla con color rojo (oscuro) y escribe "Prueba" con letras blancas al principio de la línea 20 de pantalla.*

14. Más Control Del Teclado

Hemos visto que podemos esperar a que el usuario teclee algo, utilizando la orden INPUT. Pero esta orden tiene una característica que hace que no resulte útil en muchas ocasiones: el programa queda**totalmente parado** hasta que no se teclee la respuesta y se pulse la tecla Intro.

Eso es algo que en muchas ocasiones no nos interesará. Por ejemplo, si queremos crear un juego, no podemos hacer que todos los enemigos queden parados hasta que nosotros pulsemos una tecla. El programa debería seguir avanzando, comprobar cada cierto tiempo si hemos pulsado un tecla, y proseguir después, tanto si la hemos pulsado o no.

Casi todos los BASIC incorporan una forma de comprobar si se ha pulsado una tecla, que suele ser empleando la orden INKEY$. Ésta nos devolverá la tecla que se ha pulsado, o un texto vacío ("", dos comillas dobles seguidas, sin ningún espacio entre ellas) si no se ha pulsado ninguna tecla. Un primer ejemplo de su uso podría ser este:

```
10 tecla$ = INKEY$
20 IF tecla$ = "" THEN GOTO 10
30 PRINT "Ha pulsado la tecla "; tecla$
```

(Fuente "1401.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC)

Los valores que devuelve cada tecla pulsada los podemos prever en la mayoría de los casos: Será "a" si se pulsa la tecla de la A y no están las mayúsculas activadas, y será "A" si se pulsa la tecla de la A y mientras están las mayúsculas activadas (con la tecla Bloq Mays o manteniendo pulsada la tecla de mayúsculas mientras se pulsa A); de igual modo obtendremos un "1" si se pulsa la tecla del 1, tanto en el teclado alfanumérico como en el teclado numérico auxiliar. Pero hay ciertas teclas que no son símbolos alfanuméricos y no resulta tan evidente el valor que devolverán. Vamos a ver alguna de ellas:

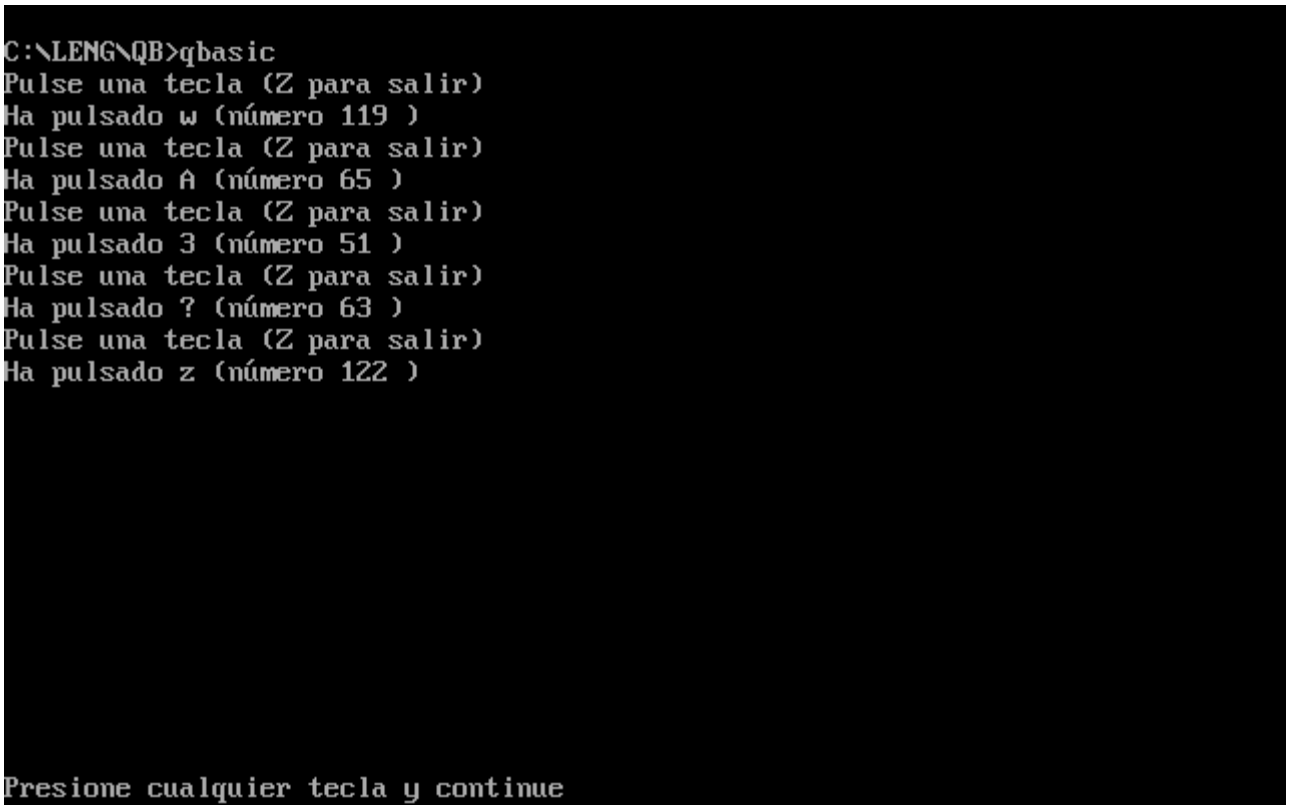
- Intro devuelve el carácter número 13.
- ESC devuelve el carácter número 27.
- Retroceso devuelve el carácter número 8.
- El "Tabulador" devuelve el carácter número 9.
- Las teclas de función F1 a F12 de un PC, así como otras teclas especiales de estos mismos ordenadores, devuelven dos bytes seguidos: primero un 0 y luego otro código que es el número de tecla que se ha pulsado (desde F1=59 hasta F10 =68, y también 133 para F11 y 134 para F12).

Podemos crear nosotros mismos un programa que nos muestre el símbolo que corresponde a cada tecla y el número que ese símbolo tiene dentro del código ASCII (que ya comentamos en el apartado 10:

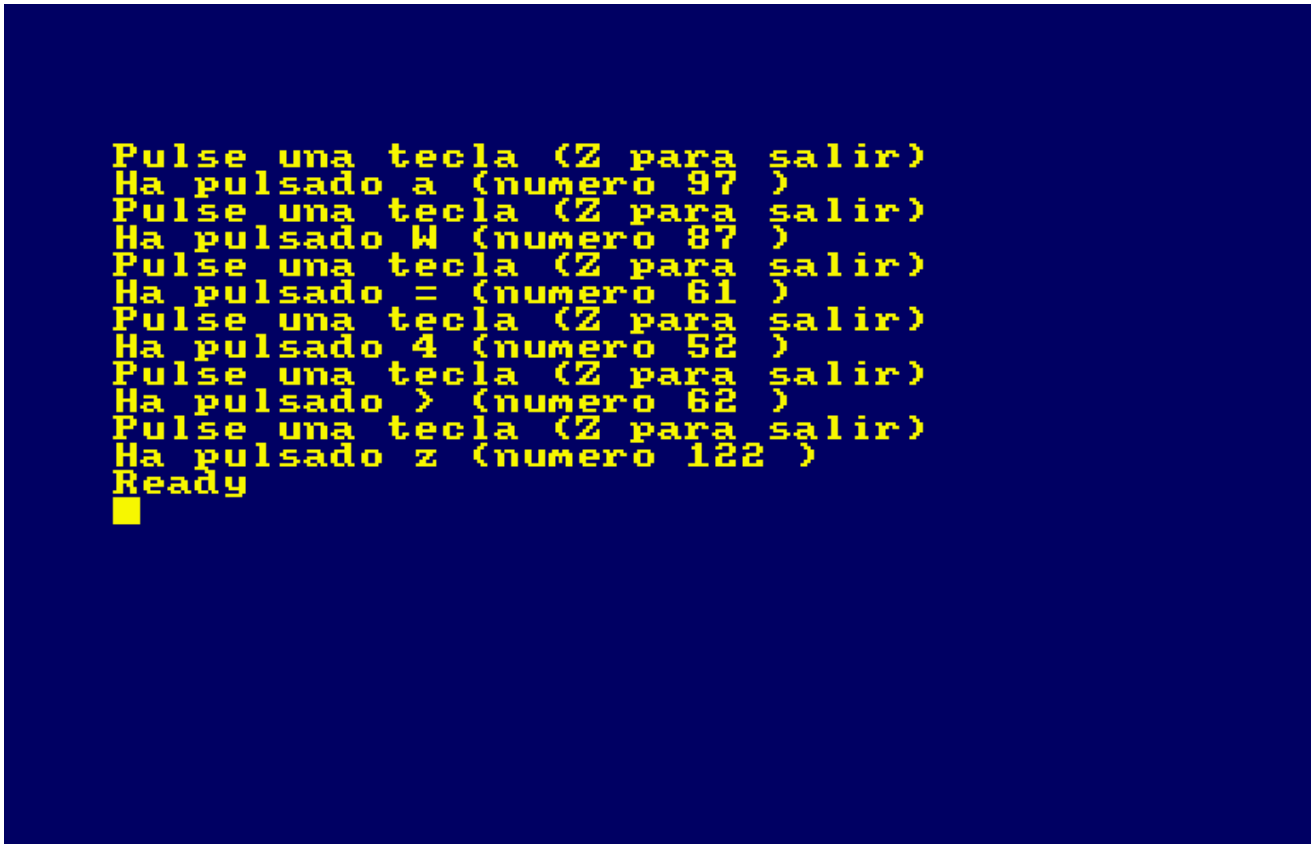
```
10 PRINT "Pulse una tecla (Z para salir)"
20 tecla$ = INKEY$
30 IF tecla$ = "" THEN GOTO 20
40 PRINT "Ha pulsado "; tecla$; " (numero"; ASC(tecla$); ")"
50 IF (tecla$ = "z") OR (tecla$ = "Z") THEN END
60 GOTO 10
```

(Fuente "1402.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC)

Que daría este resultado con QBasic



Y este en un Amstrad CPC



En el caso de **Basic256**, no existe INKEY$, sino KEY, que nos da el número asociado a cada carácter:

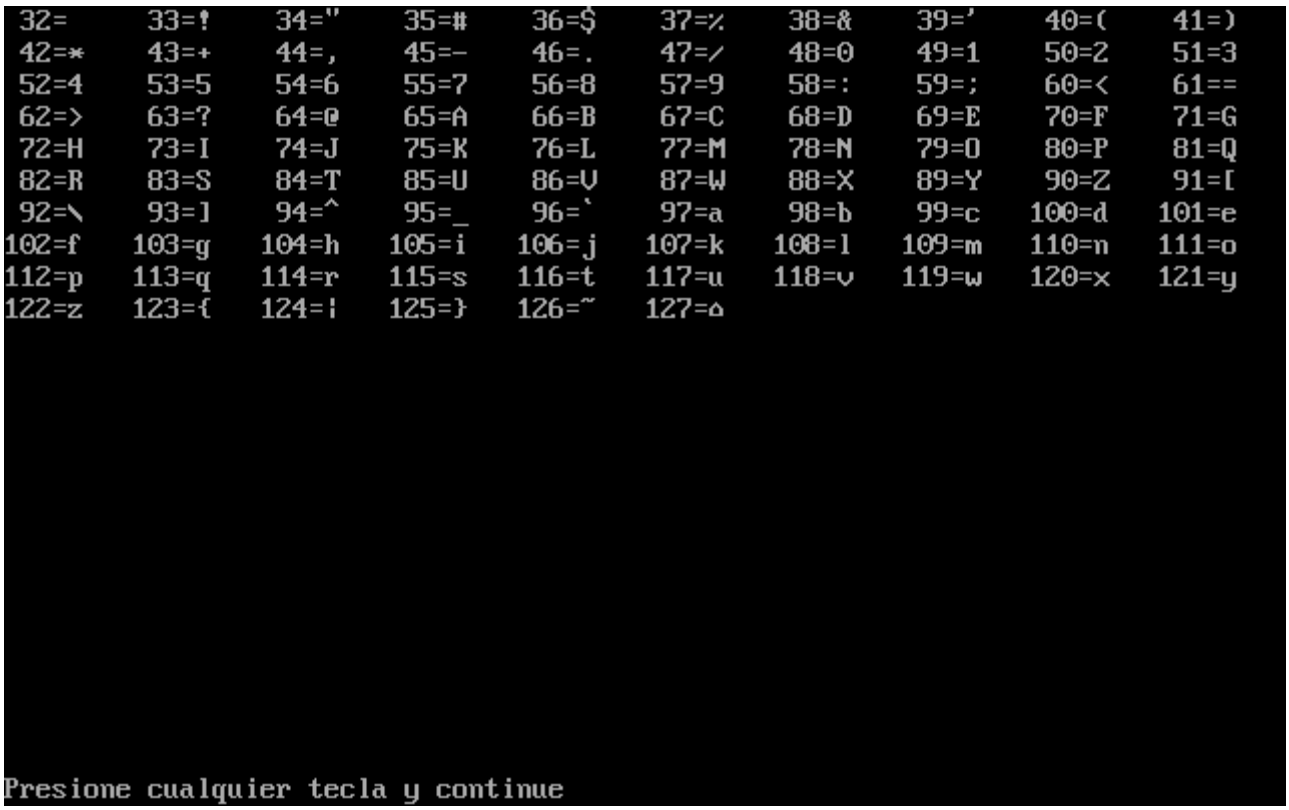
```
tecla = KEY
IF tecla = 65 THEN PRINT tecla
```

(Fuente "1403.bas". Probado en Basic256)

Para saber el número que corresponde a cada carácter deberemos conocer el código ASCII o bien crear un programa que nos muestre todos ellos:

```
1 REM Tabla ASCII
10 CLS
20 FOR caracter = 32 TO 127
30   PRINT USING "###"; caracter;
40   PRINT "="; CHR$(caracter); "   ";
50 NEXT caracter
```

(Fuente "1404.bas". Probado en GwBasic 2.02, Locomotive Basic de Amstrad CPC)



Ejercicio propuesto 14.1: *Crea un programa que se quede parado hasta que se pulse la "A".*

Ejercicio propuesto 14.2: *Crea un programa que se quede parado hasta que se pulse la tecla ESC.*

Ejercicio propuesto 14.3: *Crea un programa que te pida tu nombre, pero no lo muestre en pantalla hasta que pulse Enter (o Intro) para terminar.*

Ejercicio propuesto 14.4: *Crea un programa que te pida tu nombre, pero muestre asteriscos en lugar de cada carácter real.*

Ejercicio propuesto 14.5: *Crea un programa que te pida tu nombre, pero muestre asteriscos en lugar de cada carácter real, y que permita "retroceder" para borrar.*

Ejercicio propuesto 14.6: *Crea un programa que muestre una letra "A" en la primera columna de la fila 10 de la pantalla. Cada vez que el usuario pulse una tecla, la letra se moverá una posición hacia la derecha, hasta que el usuario pulse ESC o se llegue a la última columna de la pantalla, momento en que terminará.*

Ejercicio propuesto 14.7: *Crea un programa que muestre una letra "A" rebotando de un lado a otro de la pantalla, en horizontal (deberá moverse cada vez que el usuario pulse una tecla.*

Ejercicio propuesto 14.8: *Crea un programa que muestre un asterisco rebotando por la pantalla. Comenzará en una posición al azar, y se moverá hacia la derecha y hacia abajo cada vez que el usuario pulse una tecla. Cuando llegue al extremo derecho de la pantalla, rebotará hacia la izquierda, y cuando llegue al extremo inferior pasará a moverse hacia arriba. Debe continuar rebotando hasta que se pulse ESC.*

15. Numeración De Líneas: Auto, Renum, Edit...

En los BASIC "antiguos" (con números de línea) más completos disponíamos de una serie de ayudas a la hora de teclear nuestros programas y a la hora de intentar descubrir fallos. Vamos a ver algunas de ellas.

Una orden que podía ayudarnos cuando teníamos que teclear un programa mínimamente largo era la orden **AUTO**. Esta hacía que comenzasen a aparecer los **números de las líneas automáticamente**, de modo que nosotros sólo teníamos que centrarnos en escribir el contenido de cada orden en sí. Si solo escribíamos la orden AUTO, se comenzaría a numerar a partir de la línea 10, y se aumentaba de 10 en 10 (al pulsar Intro aparecerá la línea 20, luego la 30, y así sucesivamente):

```
AUTO
```

(Fuente "1501.bas". Probado en Locomotive Basic de Amstrad CPC)

En los BASIC que incorporaban esta posibilidad, se solía poder personalizar también un poco más. Por ejemplo, podríamos decir que queríamos comenzar a numerar a partir de la línea 200 (e incrementando de 10 en 10) así:

```
AUTO 200
```

(Fuente "1502.bas". Probado en Locomotive Basic de Amstrad CPC)

y si queremos que aumente de 5 en 5 (también a partir de la línea 200) sería:

```
AUTO 200,5
```

(Fuente "1503.bas". Probado en Locomotive Basic de Amstrad CPC)

Puede ocurrir que nos hayamos quedado sin espacio para insertar nuevas líneas. Es decir, entre la línea 10 y la 20 podemos incluir una línea 15, pero entre la línea 2 y la 3 no podemos insertar más líneas (no se permiten números de línea con decimales: no es válida la línea 2,5 ni la 2.33).

Una opción que nos permiten algunos de los mejores entre los BASIC "antiguos" es la **RENUM**, que**vuelve a numerar** las líneas de 10 en 10, y la primera pasa a ser la línea 10:

(Fuente "1504.bas". Probado en Locomotive Basic de Amstrad CPC)

Eso sí, hay que tener en cuenta que esto puede dar lugar a **problemas** si se ha empleado la orden GOTO o GOSUB: el siguiente programa

```
1 PRINT "HOLA"
2 GOTO 1
```

(Fuente "1505.bas". Probado en Locomotive Basic de Amstrad CPC)

se podría convertir en el siguiente, que es incorrecto porque ya no existe la línea 1:

```
10 PRINT "HOLA"
20 GOTO 1
```

(Fuente "1506.bas". Probado en Locomotive Basic de Amstrad CPC)

En este caso, cuando nuestro intérprete de BASIC llegue a la línea 2, aparecería un mensaje de error que diría que no existe la línea 1, pero en casos como el siguiente puede ser aún más grave:

```
5 X = 1
20 GOTO 40
40 X = X + 1
120 PRINT X
```

(Fuente "1507.bas". Probado en Locomotive Basic de Amstrad CPC)

Al cambiar la numeración de las líneas, **cambiará la estructura del programa**, la nueva línea 30 (que antes era la 40) es ignorada, sin ningún tipo de mensaje de error, con lo que el programa muestra en pantalla el valor 1 en vez de 2 (ha cambiado la lógica del programa sin darnos cuenta):

```
10 X = 1
20 GOTO 40
30 X = X + 1
40 PRINT X
```

(Fuente "1508.bas". Probado en Locomotive Basic de Amstrad CPC)

Para evitar estos problemas, la orden RENUM no se debería utilizar, salvo que sea estrictamente imprescindible, y en este caso deberíamos imprimir el programa inicial y el modificado, y compararlos para asegurarnos de todos los saltos sigan llegando al punto al que deben (Aun así, no siempre todo es tan crítico: algunas versiones de BASIC, como la de los Amstrad CPC, **suelen corregir los números de línea** que aparecen dentro de cada orden, con lo que es habitual que no se altere la lógica del programa).

Para **corregir** una línea ya escrita en un intérprete de BASIC clásico, normalmente tendremos 3 opciones:

- La que "nunca falla" es volver a escribirla.
- Algunos intérpretes permiten que, si esa línea está a la vista en pantalla, usemos las "flechas" del teclado para desplazarnos hasta ella, corrijamos las letras que nos interesen, y al pulsar la tecla Intro, la línea quedará actualizada.
- Otros muchos permiten escribir **EDIT** seguido del número de línea para poder modificar esa línea y corregir sólo las letras o símbolos que sean incorrectos:

```
EDIT 120
```

(Fuente "1509.bas". Probado en Locomotive Basic de Amstrad CPC)

16. Gráficos

La forma de crear gráficos de alta resolución desde el lenguaje BASIC **depende mucho de la versión** del Basic que estemos empleando. Me centraré en **QBasic** para MsDos (que en su mayor parte funcionará sin cambios en GwBasic), y después comentaré los principales cambios para alguna otra versión.

16.1. Entrar a modo gráfico

Un ejemplo básico sería el que fuera capaz de **cambiar a modo gráfico y dibujar una línea**. Así, en QBasic podríamos pasar a modo gráfico de 320x200 puntos (que permitirá casi cualquier ordenador tipo PC, por antiguo que sea) y dibujar una línea desde la esquina superior izquierda hasta el centro de la pantalla

```
10 SCREEN 1
20 LINE (0, 0)-(160, 100)
```

(Fuente "1601.bas". Probado en GwBasic 2.02, QBasic 1.1)

Primero haré un par de **comentarios** sobre este programa y después veremos algo más de detalle sobre las órdenes SCREEN y LINE

- He incluido números de línea por seguir la misma "metodología" que hasta ahora, pero ya sabemos que en el caso de QBasic no es necesario indicarlos, basta con escribir las líneas una tras otra.
- La orden SCREEN es la que nos permite cambiar de un modo de pantalla a otro (texto o gráfico, con mayor o menor nivel de detalle). La orden LINE será la que usaremos para dibujar líneas en pantalla.

Los **modos de pantalla** que podemos elegir en QBasic son:

- 0: Modo sólo de texto (habitualmente 80 columnas y 25 filas).
- 1: Modo gráfico de 320 x 200 puntos. Es necesario tener una tarjeta gráfica CGA. Los colores a usar están limitados: el fondo se puede elegir entre 16 posibles, pero para primer plano se ha de escoger uno de dos juegos de 3 colores prefijados.
- 2: Modo gráfico de 640 x 200 puntos. Es necesaria una tarjeta gráfica CGA. Sólo se puede usar un color de fondo y otro de primer plano.
- 3: Modo gráfico de 720 x 348 puntos, monocromo. Es necesario tener una tarjeta gráfica Hercules y haber cargado el controlador "MSHERC.COM".
- 4: Modo gráfico de 640 x 400 puntos, fondo negro y un color de primer plano a elegir entre 16. Sólo es para ciertos ordenadores Olivetti.
- 7: Modo gráfico de 320 x 200 puntos, en 16 colores. Necesaria una tarjeta EGA.
- 8: Modo gráfico de 640 x 200 puntos, en 16 colores. Necesaria una tarjeta EGA.
- 9: Modo gráfico de 640 x 350 puntos, con 16 colores (de una paleta de 64 colores posibles). Necesaria una tarjeta EGA.
- 10: Modo gráfico de 640 x 350 puntos, monocromo. Necesaria una tarjeta EGA y monitor monocromo.
- 11: Modo gráfico de 640 x 480 puntos, 2 colores. Necesaria una tarjeta MCGA o VGA.
- 12: Modo gráfico de 640 x 480 puntos, 16 colores (de entre aproximadamente 262.000 colores posibles). Necesaria una tarjeta VGA.
- 13: Modo gráfico de 320 x 200 puntos, 256 colores (de entre aproximadamente 262.000 colores posibles). Necesaria una tarjeta MCGA o VGA.

Hoy en día, en cualquier ordenador mínimamente "moderno" (un PC con procesador 386 o superior) lo razonable es usar los **modos 12 o 13**, según deseemos tener más puntos en pantalla (que serán puntos más pequeños, tendremos más detalle) o mayor número de colores.

Algunos de estos modos de pantalla permiten trabajar con varias " páginas" de información, una de las cuales será visible en cada momento (las demás quedan "ocultas", pero se pueden mostrar cuando queramos, lo que era una forma habitual de crear animaciones). Es un detalle en el que nosotros no entraremos.

16.2. Dibujar líneas

La forma básica de crear **líneas** es: LINE (x1, y1) - (x2,y2) Es decir, se indican las coordenadas X e Y de un extremo de la línea, entre paréntesis, después un guión y después las coordenadas X e Y del otro extremo de la línea, también entre paréntesis.

También se puede añadir otro "parámetro" más, para indicar el **color** de la línea. Por ejemplo, podríamos dibujar una línea amarilla en el modo 640x480 de 16 colores, así:

```
10 SCREEN 2
20 LINE (0, 0)-(160, 100), 14
```

(Fuente "1602.bas". Probado en GwBasic 2.02, QBasic 1.1)

Pero todavía podemos hacer más cosas con la orden LINE. Por ejemplo, añadiendo el parámetro "B" podemos dibujar un **recuadro** (en inglés "box") en vez de una línea, que tenga esos dos puntos como vértices opuestos:

```
10 SCREEN 12
20 LINE (0, 0)-(160, 100), 14, B
```

(Fuente "1603.bas". Probado en QBasic 1.1)

Y si en vez de una "B" utilizamos "BF", se dibujará un **recuadro relleno** (del inglés "box" y "filled"):

```
10 SCREEN 12
20 LINE (0, 0)-(160, 100), 14, BF
```

(Fuente "1604.bas". Probado en QBasic 1.1)

Existen otras posibilidades en la orden LINE, como por ejemplo la de dibujar líneas discontinuas (según un cierto patrón que nosotros indiquemos), pero tampoco entraremos a ese nivel de detalle.

16.3. Dibujar puntos aislados

Podemos dibujar puntos aislados (útiles, por ejemplo, para hacer un fondo de estrellas básico), usando la orden PSET, a la que se le indican las coordenadas del punto y el color:

```
10 SCREEN 12
20 PSET (320, 200), 15
```

(Fuente "1605.bas". Probado en QBasic 1.1)

16.4. Dibujar círculos

También podemos dibujar **círculos**, con la orden **CIRCLE**, en cuyo formato básico se indica el centro y el radio:

```
10 SCREEN 12
20 CIRCLE (320, 200), 100
```

(Fuente "1606.bas". Probado en QBasic 1.1)

La primera mejora es indicar el color. Podemos dibujar el círculo en amarillo:

```
10 SCREEN 12
20 CIRCLE (320, 200), 100, 14
```

(Fuente "1607.bas". Probado en QBasic 1.1)

Otros tres parámetros que pueden ser interesantes son: **ángulo inicial** (en radianes), **ángulo final**(también en radianes) y **relación de aspecto** (diferencia de tamaño entre diámetros, para que el resultado sea una **elipse** en vez de un círculo):

```
10 SCREEN 12
20 CIRCLE (320, 200), 100, 0, 3.14, 0.5
```

(Fuente "1608.bas". Probado en QBasic 1.1)

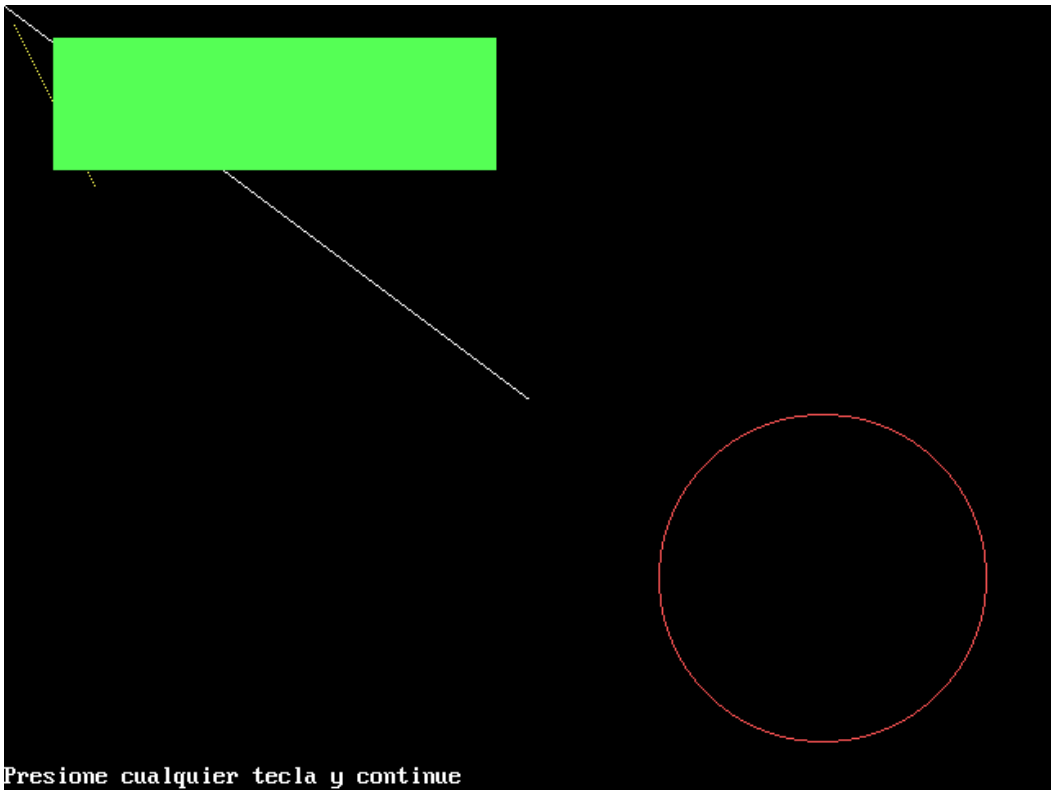
Este ejemplo dibuja media elipse, desde los 0 radianes (0 grados) hasta los 3.14 radianes (PI radianes, 180 grados), y en la que el diámetro vertical es la mitad (0.5) del horizontal.

(Se comienza a dibujar en el extremo más a la derecha del círculo -las 3 en un reloj "de agujas"-, y se avanza en sentido antihorario -contrario a las agujas del reloj-)

Y se pueden hacer **más cosas**: rellenar zonas de pantalla, cambiar la paleta de colores, usar coordenadas relativas en vez de absolutas... pero no profundizaremos tanto... al menos por ahora. Sí veremos un ejemplo básico que junto todo lo que hemos visto, para luego adaptarlo a otros sistemas:

```
10 REM Modo de pantalla 640x480, 16 colores
20 SCREEN 12
30 REM Linea blanca en la mitad superior izq
40 LINE (0, 0)-(320, 240), 15
50 REM Varios puntos amarillos cerca de esa linea
55 FOR i = 1 TO 50
60   PSET (5 + i, 10 + i * 2), 14
65 NEXT
70 REM Rectangulo verde cerca de esa linea
80 LINE (30, 20)-(300, 100), 10, BF
90 REM Circulo rojo en el otro extremo
100 CIRCLE (500, 350), 100, 12
```

(Fuente "1609.bas". Probado en QBasic 1.1)



16.5. Gráficos en un Amstrad CPC

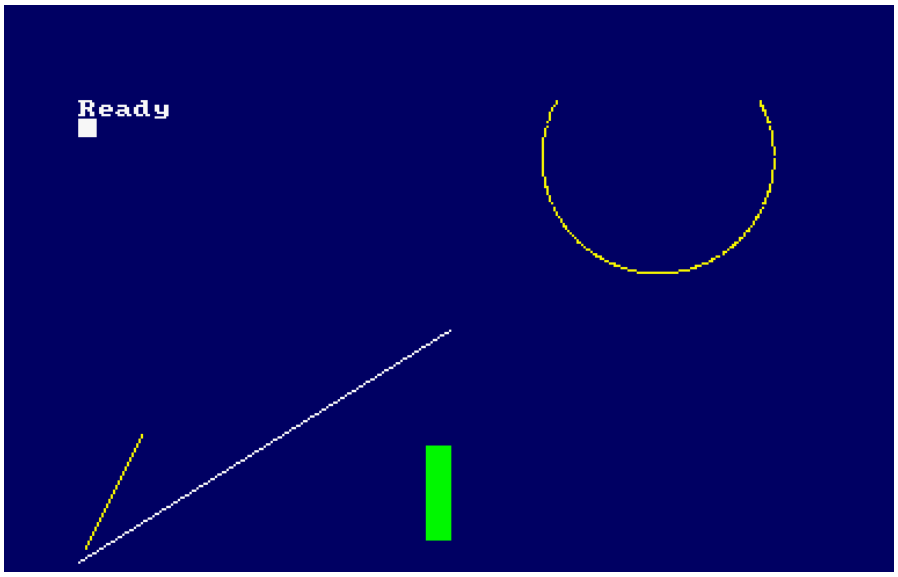
En los ordenadores clásicos Amstrad CPC las posibilidades básicas son las mismas, pero cambia la forma de conseguir las cosas. Por ejemplo:

- Hay 3 modos gráficos, con distinta resolución, pero en todos ellos las coordenadas máximas son de 640 x 400 y el centro está en (320,200)
- En el "modo 2" hay 640x200 puntos en pantalla, pero sólo se puede usar 2 colores; en el "modo 1" hay 320x200 puntos con 4 colores; en el "modo 0" hay 160x200 puntos con 16 colores.
- Con la orden "INK" se puede asignar a cada tinta uno de los 27 colores posibles de la paleta.
- "MOVE" permite movernos hasta ciertas coordenadas de pantalla. Desde ellas, podríamos dibujar una línea con "DRAW".
- Para dibujar puntos aislados se usa "PLOT".
- No existe una orden que permita dibujar rectángulos. Para un rectángulo hueco, dibujaremos las 4 líneas que forman su contorno; para uno relleno, dibujaremos líneas a lo largo de toda su superficie.
- De igual modo, en un CPC no existe orden para dibujar círculos, pero se puede usar la ecuación matemática de la circunferencia, usando "senos" y "cosenos" para calcular las coordenadas de cada punto.

Así, un programa parecido al anterior, pero adaptado a un CPC podría ser:

```
10 REM Modo de pantalla 320x200, 4 colores
20 MODE 1
25 INK 1, 26
30 REM Linea blanca en la mitad inferior izq
40 MOVE 0, 0: DRAW 320, 200, 1
50 REM Varios puntos amarillos cerca de esa linea
52 INK 2, 24
55 FOR i = 1 TO 50
60   PLOT 5 + i, 10 + i * 2, 2
65 NEXT
70 REM Rectangulo verde cerca de esa linea
75 INK 3, 18
80 FOR posicY = 20 TO 100
83   MOVE 320,posicY: DRAW 300, posicY, 3
86 NEXT posicY
90 REM Circulo amarillo en el otro extremo
100 centroX = 500: centroY = 350: radio = 100
110 FOR angulo = 1 TO 360
120   PLOT centroX + radio*COS(angulo), centroY + radio*SIN(angulo), 2
130 NEXT angulo
```

(Fuente "1610.bas". Probado en Locomotive Basic de Amstrad CPC)



Pero este programa es lento, tarda unos 14.2 segundos en dibujar esas figuras. Se puede acelerar si el rectángulo lo dibujamos trazando una de cada 2 líneas (hay 200 líneas en pantalla, aunque las coordenadas "virtuales" lleguen hasta 400) y si para el círculo no dibujamos todos los puntos sino varias líneas cortas en orden:

```
10 REM Modo de pantalla 320x200, 4 colores
20 MODE 1
25 INK 1, 26
30 REM Linea blanca en la mitad inferior izq
40 MOVE 0, 0: DRAW 320, 200, 1
50 REM Varios puntos amarillos cerca de esa linea
52 INK 2, 24
55 FOR i = 1 TO 50
60   PLOT 5 + i, 10 + i * 2, 2
65 NEXT
70 REM Rectangulo verde cerca de esa linea
75 INK 3, 18
80 FOR posicY = 20 TO 100 STEP 2
83   MOVE 320,posicY: DRAW 300, posicY, 3
86 NEXT posicY
90 REM Circulo amarillo en el otro extremo
100 centroX = 500: centroY = 350: radio = 100
105 MOVE centroX+radio, centroY
110 FOR angulo = 0 TO 360 STEP 10
120   DRAW centroX + radio*COS(angulo*3.14/180), centroY + radio*SIN(angulo*3.14/
180), 2
130 NEXT angulo
```

(Fuente "1611.bas". Probado en Locomotive Basic de Amstrad CPC)

El resultado es visualmente idéntico, pero ahora tarda 2.5 segundos, mucho más razonable (y se podría mejorar aún más, pero no afinaremos tanto).

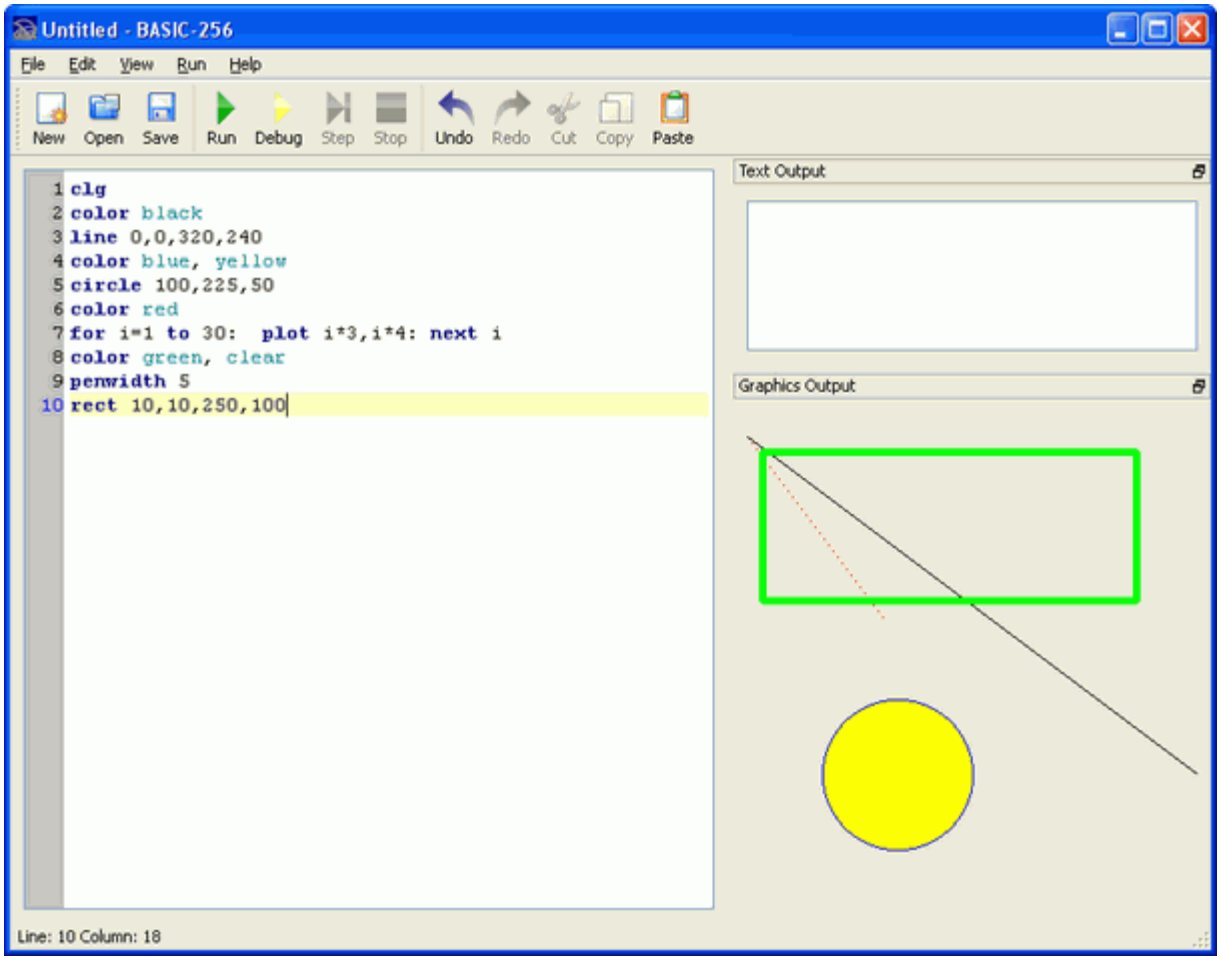
16.6. Gráficos con Basic 256

En un ordenador moderno, que use un Windows reciente o Linux, podemos usar Basic256 para crear gráficos sencillos. Las ideas básicas son:

- Se puede borrar la pantalla gráfica con "CLG"/li>
- Para dibujar líneas se usa LINE, junto con las coordenadas X e Y de origen y las X e Y de final.
- Podemos dibujar puntos con PLOT, indicando la posición X,Y del punto.
- En ambos casos, se puede indicar el color con la palabra "COLOR" seguido del nombre en inglés (por ejemplo, "COLOR BLUE")
- Para dibujar rectángulos se emplea "RECT", indicando las coordenadas de dos esquinas opuestas (superior izquierda e inferior derecha).
- Los círculos se consiguen con "CIRCLE", detallando las coordenadas del centro y el radio.
- Tanto para el rectángulo como para el círculo, podemos indicar el color de relleno, distinto del color de borde, si detallamos un segundo color en la orden "COLOR" (por ejemplo, "COLOR BLUE, YELLOW"), Si queremos que uno de los colores sea transparente, por ejemplo para hacer un círculo que no esté relleno, podemos indicar el color "CLEAR": "COLOR RED, CLEAR"
- Si queremos dibujar líneas (o contornos) de más de un punto de grosor, podemos indicar la anchura del pincel con PENWIDTH.

```
clg
color black
line 0,0,320,240
color blue, yellow
circle 100,225,50
color red
for i=1 to 30:  plot i*3,i*4: next i
color green, clear
penwidth 5
rect 10,10,250,100
```

(Fuente "1612.bas". Probado en Basic256)



Ejercicio propuesto 16.1: *Crea un programa que dibuje un triángulo cuyas esquinas estén en las coordenadas (100,50), (300,50), (210, 180).*

Ejercicio propuesto 16.2: *Crea un programa que dibuje un recuadro relleno de color rojo que ocupe toda la pantalla, excepto un margen de 20 píxeles.*

Ejercicio propuesto 16.3: *Crea un programa que dibuje 7 círculos concéntricos, imitando una diana.*

Ejercicio propuesto 16.4: *Si has estudiado trigonometría, dibuja la gráfica de la función "seno", tomando el ejemplo de la forma de dibujar un círculo en el Amstrad CPC.*

Ejercicio propuesto 16.5: *Crea un programa que dibuje un hexágono de color azul y relleno amarillo. El hexágono deberá estar en el centro de la pantalla y tener 80 píxeles de lado.*

Ejercicio propuesto 16.6: *Dibuja un campo de estrellas, formado por 500 puntos blancos en posiciones al azar.*

17. Condiciones Mejoradas: CASE

Hemos visto cómo comprobar condiciones con IF. Es algo que todos los Basic permiten, aunque algunos llegan más allá que otros, permitiendo indicar qué hacer si no se cumple la condición (ELSE).

En cualquier caso, cuando hay que comprobar **muchos posibles valores**, resulta incómodo encadenar muchos IF seguidos. Por eso, los Basic más modernos (como QBasic) incluyen la construcción "**SELECT CASE**". Ésta se usa así:

```
SELECT CASE variable
CASE valor1
lista de órdenes
CASE valor2
lista de órdenes
CASE ELSE
lista de órdenes
END SELECT
```

También se puede comparar con **varios valores**, de un determinado rango (4 TO 12) o que cumplan una determinada condición (IS >=30). Veamos un ejemplo:

```
INPUT "Escriba la nota obtenida (0-10): ", nota
SELECT CASE nota

CASE 10
  PRINT "Matricula de honor"
  PRINT "Un examen Impecable"

CASE 9
  PRINT "Sobresaliente"
  PRINT "Un examen casi perfecto"

CASE 7 TO 8
  PRINT "Notable"
  PRINT "Los fundamentos están bastante consolidados"

CASE 5 TO 6
  PRINT "Aprobado"
  PRINT "Conocimientos básicos"

CASE IS <= 4
  PRINT "Suspenso"
  PRINT "No ha adquirido los suficientes conocimientos"

END SELECT
```

(Fuente "1701.bas". Probado en QBasic 1.1)

(Como se ve en este ejemplo, podemos tabular un poco más a la derecha los pasos que se debe dar en cada caso, para que el programa sea más legible.

Ejercicio propuesto 17.1: *Crea un programa que pida una letra al usuario y le diga si es una vocal o alguna otra letra.*

Ejercicio propuesto 17.2: *Busca información sobre el sistema de notas norteamericano (en el que "A" representa la máxima nota, "B" es una nota inferior a ella y así sucesivamente) y crea un programa que convierta de ese sistema de notas a una valoración numérica del 1 al 100.*

Ejercicio propuesto 17.3: *Crea un programa que muestre un menú al usuario con las opciones "1.- Añadir datos", "2.- Buscar datos", "3.- Modificar datos", "4.- Borrar datos". Según la opción que escoja el usuario, se le responderá con un mensaje adecuado, como por ejemplo "Ha escogido Buscar".*

18. Tipos de variables.

Existen versiones antiguas del lenguaje BASIC que eran capaces de manejar muy pocos tipos de datos. Por ejemplo, la primeras sólo podían manejar números enteros (sin decimales), mientras que algunas ligeramente más modernas permiten números enteros y reales (con decimales, hasta una cierta precisión), y la mayoría permiten también cadenas de texto de longitud variable. En versiones modernas es habitual que podamos incluso definir nuestros propios tipos de datos, como veremos dentro de poco.

Cuando una versión de BASIC permitía manejar distintos tipos de datos, el tipo de la variable se indicaba añadiendo un sufijo al nombre de la variable, un símbolo especial. Es algo que ya hemos utilizado: hemos visto que si al final de una variable escribimos un símbolo $, indicamos que queremos que esa variable almacene textos en vez de números:

```
10 LET NOMBRE$="JUAN PEDRO"
20 PRINT "HOLA " ; NOMBRE$
```

(Fuente "1801.bas". Probado en Locomotive Basic de Amstrad CPC)

De igual manera, en la gran mayoría de las versiones de BASIC se puede posponer el prefijo % para indicar que una variable que almacene números va a usarse exclusivamente para números enteros (sin decimales). Esto tiene su utilidad: un número sin decimales normalmente requerirá menos espacio en memoria que un número real, y además las operaciones sobre él se realizarán con mucha mayor rapidez.

Por ejemplo, si tenemos que hacer una operación 200 veces, puede ser mucho más rápido obligar al ordenador a trabajar con enteros si realmente no necesitamos decimales. Podemos sumar los 200 primeros números enteros así:

```
10 suma% = 0
20 FOR i% = 1 TO 200
30   suma% = suma% + i%
40 NEXT i%
```

(Fuente "1802.bas". Probado en Locomotive Basic de Amstrad CPC)

En un ordenador Amstrad CPC, con procesador a 4 MHz, este programa tarda en ejecutarse 0.39 segundos, frente a los 0.60 segundos que tarda este programa, que es casi idéntico, con la única diferencia de que no hemos especificado que queremos trabajar con números enteros, así que se da por sentado que se trata de números reales.

```
10 suma = 0
20 FOR i = 1 TO 200
30   suma = suma + i
40 NEXT i
```

(Fuente "1803.bas". Probado en Locomotive Basic de Amstrad CPC)

Se trata de una mejora de velocidad cercana al 33%. Con operaciones más complicadas que la suma, la diferencia puede ser aún mayor: el siguiente programa, que hace una multiplicación sencilla, tarda 22.44 segundos (en un CPC a 4 MHz)

```
10 FOR i = 1 TO 5000
20   resultado = i * 5 + 1
30 NEXT i
```

(Fuente "1804.bas". Probado en Locomotive Basic de Amstrad CPC)

Mientras que la versión con números enteros tarda 12.21 segundos, casi la mitad:

```
10 FOR i% = 1 TO 5000
20   resultado% = i% * 5 + 1
30 NEXT i%
```

(Fuente "1805.bas". Probado en Locomotive Basic de Amstrad CPC)

En otras versiones de BASIC más modernas, se podía indicar explícitamente el tipo de una variable, y así no era necesario añadir siempre esos símbolos %. Por ejemplo, con **DEFINT** definimos una variable como "entero", de modo que el programa anterior se podría escribir también así:

```
10 DEFINT suma
20 DEFINT i
10 suma = 0
20 FOR i = 1 TO 5000
30   suma = suma + i
40 NEXT i
```

(Fuente "1806.bas". Probado en Locomotive Basic de Amstrad CPC)

(Nota: la mayoría de los BASIC que permiten DEFINT también permiten indicar varias variables seguidas, de modo que podíamos haber abreviado las dos primeras líneas en una que dijese DEFINT i, suma).

Una consideración más sobre DEFINT: en algunas versiones de BASIC, el hecho de escribir "DEFINT i" hará que tanto la variable "i" como cualquier otra cuyo nombre empiece por la letra "i" serán consideradas números enteros. De hecho, una forma habitual de indicar que el programa iba a trabajar sólo con números enteros era que su primera línea fuera DEFINT a-z (definir como enteros las variables cuyos nombres comiencen entre la A y la Z).

Por otra parte, en QBasic y otros BASIC modernos también se puede forzar a que una o varias variables sean consideradas de otros tipos de datos: enteros largos (DEFLNG), reales de "simple precisión" (DEFSNG), reales de "doble precisión" (DEFDBL) o cadenas de texto (DEFSTR). Sin entrar en detalles (que dependerían de cada BASIC concreto, podemos esperar que un "entero largo" permita más o menos el doble de cifras que un "entero normal" y que un "real de simple precisión" permitirá la mitad de cifras que un "real de doble precisión"). En cualquier caso, estos tipos de datos también tienen su "sufijo" asociado:

%	entero
&	entero largo
!	real de simple precisión
#	real de doble precisión
$	cadena de caracteres

Estos "sufijos" siempre tendrán prioridad sobre las órdenes DEF. Por ejemplo, en el siguiente programa definimos como enteras las variables que empiezan por "N", pero después definimos como real una llamada "numeroReal" simplemente incluyendo el sufijo "!":

```
10 DEFINT n
20 num1 = 5
30 numeroReal! = 5 / 2
```

(Fuente "1807.bas". Probado en Locomotive Basic de Amstrad CPC)

Ejercicio propuesto 18.1: *Crea un programa que te pida dos números enteros (debes declararlos como tal) y calcule su suma y su producto.*

Ejercicio propuesto 18.2: *Crea un programa que te pida dos números reales (debes declararlos como tal) y calcule su división.*

Ejercicio propuesto 18.3: *Crea un programa que te pida dos números enteros largos (si tu versión de BASIC lo permite; si no, basta con que sean "enteros normales") y calcule su división entera y el resto de esa división (pista: algunas versiones permiten calcular la división entera usando una barra invertida ("\"); en otras tendrás que truncar el resultando de la división para quitarle los decimales (usando órdenes como TRUNC o INT).*

19. Ficheros

19.1. Nociones básicas

El manejo de ficheros también es algo que varía mucho de una versión a otra, así que veremos sólo un ejemplo sencillo que nos ayude a entender los fundamentos.

Emplearemos QBasic, que permite más posibilidades que otros BASIC más antiguos.

Los **pasos básicos** para acceder a un fichero serán;

- Abrir el fichero.
- Leer datos o guardarlos.
- Cerrar el fichero.

Hay básicamente dos formas de acceder a un fichero: de modo **secuencial** (leer un dato tras otro, en "secuencia") o de modo directo (o **aleatorio**, pudiendo acceder directamente a cualquier posición del fichero).

19.2. Acceso secuencial a un fichero

Primero veremos la forma de acceder a un fichero secuencial: abriremos un fichero para escritura y guardaremos texto en él; después lo cerraremos y lo volveremos a abrir, esta vez para lectura, y mostraremos en pantalla todo su contenido

```
REM Primero vamos a crear el fichero
OPEN "TEXTO.TXT" FOR OUTPUT AS #1
WRITE #1, "Esto es la primera linea del fichero"
WRITE #1, "Segunda línea"
WRITE #1, "Tercera y última línea"
CLOSE #1

REM Ahora vamos a leer del fichero
PRINT "Contenido del fichero:"
OPEN "TEXTO.TXT" FOR INPUT AS #1
DO WHILE NOT EOF(1)
LINE INPUT #1, linea$
PRINT linea$
LOOP
CLOSE #1
PRINT "Final del fichero"
```

(Fuente "1901.bas". Probado en QBasic 1.1)

Casi es autoexplicativo:

- Abrimos el fichero con **OPEN**, indicando el nombre de dicho fichero ("TEXTO.TXT"), si lo abrimos para escribir (**FOR OUTPUT**) o para leer (**FOR INPUT**) y el número de fichero que le queremos asociar. En nuestro ejemplo, como es el único fichero que vamos a usar, le asignamos el número 1.
- Después escribimos datos en ese fichero (**WRITE** #1).
- Finalmente cerramos el fichero (**CLOSE** #1).
- En el caso de la lectura, leemos todo su contenido, hasta llegar al final del fichero 1 (**WHILE NOT EOF**(1)). Cada línea se leerá con la orden **LINE INPUT** (en nuestro caso, del fichero 1: LINE INPUT #1).

Cuando abrimos para escribir (FOR OUTPUT), si el fichero ya existiera, se borraría. Si queremos añadir en un fichero de texto que ya existe, en vez de crear uno nuevo, deberemos usar **FOR APPEND**, así:

```
OPEN "FICHERO2.TXT" FOR APPEND AS #3
```

Ejercicio propuesto 19.2.1: *Crea un programa que pida frases al usuario y las guarde en un fichero de texto, cada frase en una línea.*

Ejercicio propuesto 19.2.2: *Crea un programa que pida al usuario el nombre de un fichero de texto y muestre todo su contenido en pantalla. Tras cada 24 líneas, deberá hacer una pausa hasta que el usuario pulse Intro.*

Ejercicio propuesto 19.2.3: *Crea un programa que pida al usuario el nombre de un fichero de texto y muestre en pantalla la cantidad de líneas de texto que contiene.*

Ejercicio propuesto 19.2.4: *Crea un programa que pida al usuario el nombre de un fichero de texto, cuente cuántas líneas contiene, cree un array y guarde todas ellas, para luego mostrarlas en pantalla en orden contrario (de la última a la primera).*

Ejercicio propuesto 19.2.5: *Crea un programa que pida al usuario el nombre de un fichero de texto, guarde todas sus líneas en un array y las vuelque a un nuevo fichero llamado "salida.txt", en orden contrario (de la última a la primera).*

19.3. Acceso aleatorio a un fichero

Si queremos abrir el fichero para acceder directamente a cualquier " registro", tendremos que indicar que será de **acceso directo**.

Vamos a verlo con un ejemplo, en el que casi todo es ya conocido, salvo que cambia un poco la forma de abrir el fichero (se indica que es acceso aleatorio con **FOR RANDOM**, y se indica también la longitud de cada ficha), y que los datos se leen con **GET** y se escriben con **PUT** , indicando en cada caso el número de registro a leer o escribir.

```
REM Definimos el tipo de datos que vamos a almacenar
TYPE TipoAlumnos
  nombre AS STRING * 30
  edad AS INTEGER
  nota AS SINGLE
END TYPE

REM Reservamos espacio en memoria para 20 alumnos
DIM Alumno(1 TO 20) AS TipoAlumnos

REM Y también para una ficha temporal
DIM AlumnoTemp AS TipoAlumnos

REM Abro el fichero y guardo 10 datos en él
OPEN "ALUMNOS.DAT" FOR RANDOM AS #1 LEN = LEN(TipoAlumnos)
FOR i = 1 TO 10
  PUT #1, i, Alumno(i)
NEXT i
CLOSE #1

REM Ahora abro y leo la 5ª ficha
OPEN "ALUMNOS.DAT" FOR RANDOM AS #1 LEN = LEN(TipoAlumnos)
GET #1, 5, AlumnoTemp
PRINT "Nombre del alumno:", AlumnoTemp.nombre
PRINT "Nota del alumno:", AlumnoTemp.nota
CLOSE #1
```

(Fuente "1902.bas". Probado en QBasic 1.1)

Ejercicio propuesto 19.3.1: *Crea una pequeña agenda, que permita guardar el nombre, la edad, el teléfono y el correo electrónico de hasta 200 amigos. Deberá mostrar un pequeño menú que permita añadir un nuevo dato o ver todos los existentes.*

Ejercicio propuesto 19.3.2: *Mejora la agenda (21.3.1) para que permita buscar personas que contengan un cierto texto.*

20. Definición De Funciones Y Procedimientos

La **programación estructurada** procura que los programas sean más fáciles de diseñar y mantener. Esto se consigue mediante ciertos tipos de instrucciones, que permiten que un programa no sea simplemente una serie de instrucciones que se procesan una tras otra.

Algunas de estas instrucciones ya las hemos visto: WHILE, LOOP y UNTIL. Pero hay más posibilidades, como la de definir "bloques" dentro de un programa, empleando lo que llamaremos funciones y procedimientos.

Ya hemos empleado alguna de las "funciones" predefinidas que incorpora el lenguaje Basic, y que nos permiten calcular el valor absoluto de un número, el seno o el coseno de un ángulo, etc. Ahora comenzaremos por ver la forma en que en muchos Basic antiguos podíamos definir nuestra propias **funciones**. Vamos a crear una función que nos devuelve el cuadrado de un cierto número

```
10 DEF FNcuadrado (x) = x * x
20 PRINT "El cuadrado de 2 es "; FNcuadrado(2)
```

(Fuente "2001.bas". Probado en Locomotive Basic de Amstrad CPC)

Es decir, se usa la orden **DEF FN**, y justo a continuación de la palabra FN se indica el nombre que tendrá la función, los parámetros que aceptará esa función (en nuestro caso, un único número, que hemos llamado X, y que es el número cuyo cuadrado queremos elevar), y finamente un signo = y la expresión matemática que indica cómo calcular esa función (en nuestro caso, al ser el cuadrado, multiplicamos el número por sí mismo). En la siguiente línea hemos utilizado esa función para calcular el cuadrado del número 2 (hay que indicar el nombre de la función, también precedido por las letras FN).

Pero los Basic más modernos, como **QBasic**, las funciones pueden **dar más pasos**, en vez de limitarse a una única operación matemática, y se definen de una forma distinta:

```
FUNCTION RaizCubica (x)
ValorTemporal = ABS(x) ^ .33333333333#
IF x < 0 THEN ValorTemporal = ValorTemporal * (-1)
RaizCubica = ValorTemporal
END FUNCTION

PRINT "La raiz cúbica de -27 es "; RaizCubica(-27)
```

(Fuente "2002.bas". Probado en QBasic 1.1)

En este ejemplo, si X es negativo, QBasic da un error al elevar X a .3333, de modo que lo que he hecho es utilizar un valor temporal, elevando el valor absoluto del número y cambiando luego el signo si corresponde. Finalmente, doy a "RaizCubica" el valor definitivo.

Finalmente, en QBasic también existe la posibilidad de crear "**subrutinas**" (lo que en otros lenguajes de programación se llama "**procedimientos**"). Es algo parecido a lo que conseguíamos con GOSUB, pero de forma más elegante y más legible, además de que se pueden indicar parámetros, igual que hacíamos en las funciones. Por ejemplo, podemos crear una subrutina que escriba un cierto texto centrado en una fila de la pantalla que indiquemos:

```
SUB EscribeCentrado (texto$, y)
LOCATE y, 40 - (LEN(texto$) / 2)
PRINT texto$
END SUB

CLS
PRINT "Hola"
EscribeCentrado "Ejemplo", 10
EscribeCentrado "Segundo ejemplo", 15
```

(Fuente "2003.bas". Probado en QBasic 1.1)

Aquí se ve como la forma de usar una subrutina en QBasic también es ligeramente distinta a la de las funciones, porque en las subrutinas los parámetros no se indican entre paréntesis (además del hecho de que una función devuelve un valor, que deberemos mostrar con PRINT o utilizar como valor intermedio para otras operaciones, mientras que una subrutina no devuelve ningún valor).

También existe la posibilidad de acceder a la subrutina con la orden **CALL** (entonces sí que habrá que indicar los parámetros entre paréntesis), así:

```
SUB EscribeCentrado (texto$, y)
LOCATE y, 40 - (LEN(texto$) / 2)
PRINT texto$
END SUB

CLS
PRINT "Hola"
CALL EscribeCentrado ("Ejemplo", 10)
CALL EscribeCentrado ("Segundo ejemplo", 15)
```

(Fuente "2004.bas". Probado en QBasic 1.1)

Ejercicio propuesto 20.1: *Crea una función SUMA, que devuelva la suma de los 2 valores que se le indiquen.*

Ejercicio propuesto 20.2: *Crea una función HIPOTENUSA, que devuelva el valor de la hipotenusa de un triángulo rectángulo a partir de los valores de los 2 catetos, que se le indicarán como parámetros.*

21. Varias Variables Del Mismo Tipo: Arrays

21.1. Uso básico de arrays

Nos puede interesar tener una colección de variables del mismo tipo. Por ejemplo las edades de 20 amigos. Para ello se usa la orden DIM:

```
DIM edad(20)
```

Este tipo de datos "agrupados" es lo que se suele conocer como un "**array**", una "**matriz**", un "**vector**" o un "**arreglo**".

Se accedería a la primera de las edades con órdenes como "PRINT edad(1)", y la última como "PRINT edad(20)".

Aquí hay otra **peculiaridad** que comentar: algunas versiones de BASIC empiezan a numerar en 0, de modo que el primer elemento sería "edad(0)" y el último sería "edad(19)". En algunas versiones, como QBasic, se puede especificar si queremos empezar a numerar desde 0 o desde 1, con la orden OPTION BASE 0 o con OPTION BASE 1, respectivamente.

Podemos pedir los valores de 10 numeros, guardarlos en un array y luego mostrarlos, haciendo:

```
10 DIM dato(10)
20 REM Pedimos datos
30 FOR i=1 TO 10
40   PRINT "Dime el dato ";i
50   INPUT dato(i)
60 NEXT i
70 REM Mostramos datos
80 FOR i=1 TO 10
90   PRINT dato(i)
100 NEXT i
```

(Fuente "2101.bas". Probado en Locomotive Basic de Amstrad CPC)

Ejercicio propuesto 21.1.1: *Crea un programa que pida al usuario 10 números y luego los muestre en orden contrario al que se introdujeron.*

Ejercicio propuesto 21.1.2: *Crea un programa que pida al usuario 10 números y luego muestre su suma en una línea y, en otra línea, todos los datos que se habían introducido.*

Ejercicio propuesto 21.1.3: *Crea un programa que pida al usuario 10 números y luego calcule su media y muestre los que están por encima de la media.*

Ejercicio propuesto 21.1.4: *Crea un programa que pida al usuario 10 números y luego pregunte al usuario qué dato quiere buscar, y deberá decir si ese dato estaba entre los 10 iniciales o no.*

Ejercicio propuesto 21.1.5: *Crea un programa que pida al usuario 10 números y luego diga cuál es el mayor de los 10 (pista: toma el primer valor como "mayor provisional"; a partir de ahí, deberás comparar todos los demás valores con ese "mayor provisional"; si un valor es mayor que el "mayor provisional", ese deberá pasar a ser el nuevo "mayor provisional"; al terminar la pasada por todos los números, habrás obtenido el mayor de todos ellos).*

En QBasic también podemos indicar los valores máximo y mínimo de los índices del array:

```
DIM edad(1 TO 20)
```

de modo que el programa anterior podría quedar

```
10 DIM dato(1 TO 10)
20 REM Pedimos datos
30 FOR i=1 TO 10
40   PRINT "Dime el dato ";i
50   INPUT dato(i)
60 NEXT i
70 REM Mostramos datos
80 FOR i=1 TO 10
90   PRINT dato(i)
100 NEXT i
```

(Fuente "2102.bas". Probado en QBasic 1.1)

21.2. Arrays multidimensionales

Y también se pueden usar "varias dimensiones". Por ejemplo, si tenemos 20 deportistas (o menos) participando en 5 competiciones distintas, podríamos guardar en un único "bloque de memoria" todas las puntuaciones obtenidas por ellos, haciendo:

```
DIM puntuacion(20, 5)
```

y accederíamos a la puntuación del deportista número 10 en la competición 3 escribiendo cosas como

```
PRINT puntuacion ( 10, 3 )
```

En las versiones más modernas de BASIC podremos usar más de dos dimensiones (quizá no en algunas versiones de los años 80), de modo que podríamos hacer cosas más elaboradas, como almacenar la puntuación de 20 personas en 7 competiciones, cada una de las cuales tiene 12 pruebas, y en las que cada prueba tiene un máximo de 3 intentos:

```
DIM puntuacion(20, 7, 12, 3)
```

Una versión ampliada del primer ejercicio, con un array bidimensional formado por tres bloques de 5 elementos cada uno, podría ser:

```
10 DIM dato(5,3)
20 REM Pedimos datos
30 FOR bloque=1 TO 3
40   FOR posicion=1 TO 5
50     PRINT "Dime el dato ";posicion; "del bloque ";bloque
60     INPUT dato(posicion, bloque)
70   NEXT posicion
80 NEXT bloque
90 REM Mostramos datos
100 FOR bloque=1 TO 3
110   FOR posicion=1 TO 5
120     PRINT dato(posicion, bloque)
130   NEXT posicion
140 NEXT bloque
```

(Fuente "2103.bas". Probado en Locomotive Basic de Amstrad CPC)

Ejercicio propuesto 21.2.1: *Crea un programa que pida al usuario 2 series de 6 números y luego los muestre la media de cada serie y la media global. Debes usar un array de 2 dimensiones.*

21.3. Arrays de otros tipos de datos

Por supuesto, no sólo podemos guardar número enteros. Los arrays también se pueden usar para guardar otros tipos de variables. Por ejemplo, podemos guardar los nombres de esos 20 amigos (cadenas de caracteres) con

```
DIM nombre$(20)
```

o las estaturas de 20 personas, en metros (por lo que puede haber decimales decimales) forzando a que sean números reales, con

```
DIM estatura!(20)
```

o dejar claro que los datos van a ser números enteros con

```
DIM puntos%(20)
```

y en todas las partes del programa en las que usáramos la variable "puntos", debería aparecer terminada en el símbolo de porcentaje, así:

```
10 DIM puntos%(5)
20 REM Pedimos algunos datos
30 FOR i=1 TO 4
40   PRINT "Dime el dato ";i
50   INPUT puntos%(i)
60 NEXT i
70 REM Mostramos datos
80 PRINT "El segundo dato es ";puntos%(2)
```

(Fuente "2104.bas". Probado en Locomotive Basic de Amstrad CPC)

Ejercicio propuesto 21.3.1: *Crea un programa que pida al usuario 10 nombres y luego los muestre en orden contrario al que se introdujeron.*

Ejercicio propuesto 21.3.2: *Crea un programa que pida al usuario 8 números enteros y luego muestre el mayor de ellos (mira el ejercicio 21.1.5 si tienes dudas sobre cómo calcular el mayor).*

21.4. Valores predefinidos: READ, DATA, RESTORE.

Si declaramos una matriz con varios datos, como DIM edad(20), podríamos darles valores haciendo algo como

```
10 DIM edad(20)
20 edad(1) = 15 : edad(2) = 16 : edad(3) = 15 : edad(4) = 14
30 edad(5) = 15 : edad(6) = 15 : edad(7) = 14 : edad(8) = 16
40 edad(9) = 16 : edad(10) = 16 : edad(11) = 15 : edad(12) = 15
50 edad(13) = 14 : edad(14) = 14 : edad(15) = 15 : edad(16) = 16
60 edad(17) = 15 : edad(18) = 15 : edad(19) = 17 : edad(20) = 15
```

(Fuente "2105.bas". Probado en Locomotive Basic de Amstrad CPC)

pero esto es lento e incómodo. En su lugar, la mayoría de las versiones de BASIC incluyen la orden **DATA**, con la que se enumeran todos estos posibles valores, y la orden **READ** para leer dichos valores, de modo que podríamos conseguir el mismo efecto así:

```
10 DIM edad(20)
20 DATA 15,16,15,14, 15,15,14,16, 16,16,15,15, 14,14,15,16, 15,15,17,15
30 FOR i = 1 TO 20: READ edad(i): NEXT i
```

(Fuente "2106.bas". Probado en Locomotive Basic de Amstrad CPC)

Si en un mismo programa tenemos muchas líneas DATA distintas, que usemos desde distintas partes del programa, podemos usar la orden **RESTORE** para indicarle desde qué línea queremos leer los datos cada vez que nos interese, así:

```
10 DIM edad(20)
20 REM Datos de otro tipo
30 DATA 1125, 2450
40 RESTORE 70: REM Queremos leer las edades
50 FOR i = 1 TO 20: READ edad(i): NEXT i
60 REM Estos son los datos de las edades
70 DATA 15,16,15,14, 15,15,14,16, 16,16,15,15, 14,14,15,16, 15,15,17,15
```

(Fuente "2107.bas". Probado en Locomotive Basic de Amstrad CPC)

Ejercicio propuesto 21.4.1: *Crea un programa que prepare un array que almacene la información sobre cuántos días tiene cada mes del año (suponiendo que febrero siempre tiene 28). El usuario introducirá un número de mes (por ejemplo, el 4 para referirse a "abril") y se le dirá cuantos días tiene ese mes.*

Ejercicio propuesto 21.4.2: *Crea un programa que prepare un array que almacene los nombres de los meses. El usuario introducirá un número de mes (por ejemplo, el 4) y se le dirá el nombre de ese mes (por ejemplo, "abril").*

Ejercicio propuesto 21.4.3: *A partir de la estructura del ejercicio 21.4.1, crea un programa que pida al usuario un día y un mes y le diga cual es el número de ese día dentro del año. Por ejemplo, el 5 de enero sería el día 5 del año, mientras que el 3 de febrero sería el día 34 del año.*

22. Tipos De Datos Definidos Por El Usuario

En las versiones más modernas de BASIC podemos crear datos "a nuestra medida".

Como primer acercamiento, podemos usar la orden DIM también para **declarar una única variable** de un tipo "normal", por ejemplo:

```
DIM i AS INTEGER
DIM n AS LONG
DIM numero2 AS SINGLE
DIM resultado AS DOUBLE
DIM mensaje AS STRING

i = 5
n = 123456789
numero2 = 5.3
resultado = 9.765432
mensaje = "Hola"

PRINT i, n, numero2, resultado, mensaje
```

(Fuente "2201.bas". Probado en QBasic 1.1)

También podemos declarar a la vez varias variables del mismo tipo::

```
DIM x, y AS INTEGER
```

Y se puede limitar el máximo **número de letras** de una cadena de texto, así:

```
DIM mensaje AS STRING * 20
```

En cualquier caso, podemos obligar a que sea **necesario declarar** variables, y que el intérprete nos avise cuando encuentre alguna variable que no hemos declarado antes, con lo que evitamos los problemas de variables mal tecleadas, a las que el ordenador asignaría el valor 0 automáticamente, dando lugar a errores muy difíciles de detectar. Esto lo conseguimos con la orden

```
OPTION EXPLICIT
```

Con una construcción bastante parecida a DIM, podemos crear **nuestros propios tipos de datos**, formados por varios datos distintos de tipos "básicos". Para eso, emplearemos la orden TYPE. Luego declararíamos las variables con DIM::

```
TYPE TipoAlumnos
  nombre AS STRING * 30
  edad AS INTEGER
  nota AS SINGLE
END TYPE

DIM alumno AS TipoAlumnos

alumno.nombre = "Juan Perez"
alumno.edad = 16
alumno.nota = 8.2

PRINT "Nombre: "; alumno.nombre
PRINT "Edad: "; alumno.edad
PRINT "Nota: "; alumno.nota
```

(Fuente "2202.bas". Probado en QBasic 1.1)

Podríamos incluso crear un "array" para guardar varios datos de ese nuevo tipo:

```
TYPE TipoAlumnos
  nombre AS STRING * 30
  edad AS INTEGER
  nota AS SINGLE
END TYPE

DIM alumno(1 TO 20) AS TipoAlumnos

alumno(1).nombre = "Juan Perez"
alumno(1).edad = 16
alumno(1).nota = 8.2

PRINT "Nombre del primer alumno: "; alumno(1).nombre
PRINT "Edad del primer alumno: "; alumno(1).edad
PRINT "Nota del primer alumno: "; alumno(1).nota
```

(Fuente "2203.bas". Probado en QBasic 1.1)

También podemos dar valores a varios de esos datos repetitivos con una orden FOR. Por ejemplo, si todos nuestros alumnos tienen 15 años de edad, podríamos hacer

```
FOR i = 1 TO 20
Alumno(i).edad = 15
NEXT i
```

En **Visual Basic** existe otra posibilidad a la hora de crear nuestros tipos de datos, para tener que teclear menos cuando sean **estructuras repetitivas**: en vez de escribir

```
Alumno(1).nombre = "Juan Perez"
Alumno(1).edad = 16
Alumno(1).nota = 8.2
```

podemos utilizar la orden **WITH** y así eliminar la parte repetitiva:

```
WITH Alumno(1)
  .nombre = "Juan Perez"
  .edad = 16
  .nota = 8.2
END WITH
```

Ejercicio propuesto 22.1: *Crea un tipo de datos que permita guardar el nombre, la edad, el teléfono y el correo electrónico de una persona. Pide datos al usuario para un variable de ese tipo y luego muéstralos en pantalla.*

Ejercicio propuesto 22.2: *Amplía el ejercicio 22.1, para poder manejar los datos de 30 personas, usando un array. El usuario deberá poder añadir una nueva o ver los datos de todas ellas.*

Ejercicio propuesto 22.3: *Amplía el ejercicio 22.2, para que se pueda buscar una persona a partir de su nombre (completo o parte de él: puedes usar INSTR para ver si el nombre contiene el texto introducido).*