QBASIC II

TEMA 1.16
FICHEROS DE ACCESO DIRECTO O ALEATORIO

- 1.16.1 - Introducción a los ficheros directos
- 1.16.2 - Lectura y escritura en ficheros directos
- 1.16.3 - Mantenimiento de ficheros directos
  - 1.16.3.1 - Introducción al mantenimiento de ficheros
  - 1.16.3.2 - Altas. Añadir registros
  - 1.16.3.3 - Consultas. Búsqueda secuencial de registros
  - 1.16.3.4 - Modificaciones de registros
  - 1.16.3.5 - Bajas. Borrar registros
  - 1.16.3.6 - Compactación de ficheros directos
  - 1.16.3.7 - Elaboración de listados de datos
- 1.16.4 - Breve idea sobre bases de datos
- 1.16.5 - Otras instrucciones relacionadas con ficheros

1.16.1 - INTRODUCCIÓN A LOS FICHEROS DIRECTOS

Los ficheros secuenciales los usábamos de forma que era necesario leerlos línea por línea siempre desde el principio y sin saltar ninguna, y para guardarlos ocurría lo mismo. Ahora vamos a trabajar con ficheros de acceso directo o aleatorio, que estarán formados por "trozos" de igual tamaño, y por lo tanto el ordenador será capaz de saltar al registro que nosotros queramos para leerlo o escribirlo. Los registros se corresponden con la estructura de un tipo de datos definido por el usuario y para acceder a ellos únicamente se calcula su posición inicial a partir del número de registro y de su tamaño, que nos calcula la función LEN. No están separados de ninguna forma, ya no hablamos de líneas ni de caracteres de retorno de carro.

1.16.2 - LECTURA Y ESCRITURA EN FICHEROS DIRECTOS

Vamos con un ejemplo que define un tipo de datos compuesto de nombre y edad, y abre un fichero en modo RANDOM para almacenar en él los datos de personas que va escribiendo el usuario, al estilo de como lo haría una agenda.

```
TYPE TIPOPERSONA
       nombre AS STRING * 20
       edad AS INTEGER
END TYPE
DIM persona AS TIPOPERSONA
CLS
```

```
OPEN "agenda.dat" FOR RANDOM AS #1 LEN = LEN(persona)
n=0
DO
       n=n+1
       INPUT "Nombre: ", persona.nombre
       INPUT "Edad: ", persona.edad
       PUT #1, n, persona
       INPUT "¿Meter más datos (S/N)? ", respuesta$
LOOP UNTIL respuesta$="N"
CLOSE #1
PRINT n; "registros grabados en la agenda."
```

El resultado por pantalla podría ser este:

```
Nombre: Pepe
Edad: 22
¿Meter más datos (S/N)? S
Nombre: Paco
Edad: 25
¿Meter más datos (S/N)? S
Nombre: Ana
Edad: 28
¿Meter más datos (S/N)? N
3 registros grabados en la agenda.
```

Y el resultado en el sistema de archivos será que se ha creado un nuevo archivo llamado agenda.dat, si no existía de antes, que contiene los datos de tres personas. De la forma que está hecho el programa nuestros tres registros irían a parar siempre al principio del archivo. En el siguiente ejemplo veremos como leer los datos del archivo, ahora vamos a ver como funciona este programa.

- Lo primero que hacemos es definir un tipo de datos personalizado que tiene dos miembros, nombre de 20 caracteres y edad que es un entero.
- A continuación declaramos una variable como del tipo de datos que acabamos de definir y que nos va a servir como "Plantilla" para trabajar con nuestro archivo de acceso aleatorio.
- Después borramos la pantalla como siempre.
- Ahora abrimos el archivo "agenda.dat" en modo de acceso aleatorio (RANDOM), le asignamos el descriptor #1 y con la cláusula LEN le decimos cuanto largo van a ser los registros. Podíamos haberlo calculado nosotros viendo cuando ocupan todos los miembros de nuestro tipo de datos y poner simplemente el número de bytes, pero es más cómodo y más seguro que lo haga la función LEN (Se llama igual que la cláusula, pero cada cosa es diferente).
- Seguidamente ponemos un contador (n) a cero. En este caso en QBasic no hubiera sido necesario, pero es mejor acostumbrarse a hacerlo siempre.
- A continuación entramos en un bloque REPETIR, incrementamos en 1 el contador y le pedimos al usuario que escriba el nombre de la persona y después la edad y lo guardamos cada cosa en su sitio en la variable persona.
- Ahora vamos a escribir en el fichero de acceso aleatorio el registro completo, donde ya tenemos almacenado el nombre y la edad que acaba de escribir el usuario. Para hacer esto usamos la instrucción PUT a la que

le tenemos que pasar como parámetro el descriptor del archivo (el carácter # es opcional), el número de registro al que nos estamos refiriendo, para lo que usaremos el contador n, y los datos propiamente dichos que están almacenados en la variable persona.

- Preguntamos al usuario si quiere meter más datos y si pulsa la N mayúscula salimos del bucle, en caso contrario volvemos al principio donde incrementamos el contador, volvemos a llenar los miembros de la variable persona y finalmente los grabamos en la siguiente posición del archivo.
- Cuando salimos del bucle cerramos el archivo (hay que hacerlo siempre) y escribimos un mensaje en pantalla con el número de registros grabados aprovechando el valor de la variable n que usamos como contador.

Cuando abrimos un archivo en modo RANDOM si no existe se crea, y si existe no ocurre nada, ya que como veremos a continuación en él vamos a poder tanto leer como escribir registros.

Si al grabar un registro nos referimos a uno que ya existe, este se sobreescribirá entero, y si no existe el fichero aumentará de tamaño hasta permitir usar ese número de registro, añadiendo si es necesario más registros entre medio. La información que se almacenará en estos registros intermedios es indeterminada. Por ejemplo si tenemos un fichero con tres registros (1, 2 y 3) y escribimos en el registro 10 se almacenará nuestra información en el registro 10 y a la vez se crearán 6 registros más (4, 5, 6, 7, 8 y 9) llenos de información indeterminada.

El primer registro es siempre el 1 y el último posible dependerá del espacio disponible en disco.

Los ficheros que creemos de esta forma ya no los podremos editar con un editor de texto como el EDIT o el bloc de notas, ya que contienen una mezcla de letras y de valores numéricos que se pueden representar por caracteres especiales no imprimibles. También es necesario abrirlos siempre usando el mismo tipo de datos como "molde", ya que de no hacerlo así todos los registros se mezclarían y no nos serviría de nada.

Ahora vamos a ver otro ejemplo de programa que lea el fichero que acabamos de crear con el programa anterior.

```
TYPE TIPOPERSONA
       nombre AS STRING * 20
       edad AS INTEGER
END TYPE
DIM registro AS TIPOPERSONA
CLS
OPEN "agenda.dat" FOR RANDOM AS #1 LEN = LEN(registro)
numeroRegistros = LOF(#1) / LEN(registro)
FOR n = 1 TO numeroRegistros
       PUT #1, n, registro
       PRINT persona.nombre,  persona.edad
NEXT
CLOSE #1
```

El resultado por pantalla podría ser este:

Pepe 22
Paco 25
Ana 28

Y en el fichero no se produciría ningún cambio porque no hay ninguna instrucción de escritura.

Como se puede ver el programa es parecido al anterior. Hemos definido un tipo de datos que en esta ocasión se llama registro en vez de persona pero tiene exactamente la misma estructura y a continuación abrimos el fichero de la misma forma.

Podemos calcular el número de registros que tiene un fichero usando la función LEN que, como ya sabemos, nos da el tamaño del registro, y la función LOF (Length Of File) que nos devuelve el tamaño en bytes del fichero abierto cuyo descriptor le pasemos como parámetro (con o sin #) usando la fórmula "Tamaño del fichero dividido por tamaño del registro". De esta forma si el fichero tiene por ejemplo 200 bytes y cada registro mide 40 bytes obtenemos como resultado que hay 5 registros. Como no hay caracteres retornos de carro ni nada de eso las cuentas siempre salen exactas.

Una vez que sabemos cuantos registros hay podemos escribirlos en pantalla sencillamente usando un bucle FOR. Para leer información del fichero usamos la instrucción GET cuya forma de uso es similar a la de la instrucción PUT que ya hemos visto. Tras ejecutar la instrucción GET el contenido del registro queda almacenado en la estructura de la variable.

Si leemos más allá del final del fichero no ocurre nada, a diferencia de lo que pasaba con los ficheros secuenciales, pero la información que se nos da es indeterminada, en el mejor de los casos ceros y espacios en blanco. De todas formas lo correcto es calcular cual es el ultimo registro como hemos hecho antes y no leer nada más allá para evitar errores lógicos y posibles confusiones. También nos hará falta conocer siempre el número de registros para insertar los nuevos a continuación y no sobreescribiendo los ya existentes como hace nuestro primer ejemplo que no es nada práctico. Esto lo veremos en los siguientes apartados dedicados a lo que se conoce como "Mantenimiento de ficheros"

1.16.3 - MANTENIMIENTO DE FICHEROS DIRECTOS

1.16.3.1 - INTRODUCCIÓN AL MANTENIMIENTO DE FICHEROS

En el apartado anterior teníamos un programa para meter datos en un fichero, y otro para listar todos los datos que acabábamos de meter, y para complicarlo todo si ejecutábamos otra vez el primer programa como siempre empezábamos por el registro 1 seguro que sobreescribíamos datos. También para borrar información sólo podíamos borrar todos los datos a la vez borrando el fichero desde el sistema operativo y empezando con uno nuevo vacío.

Esta sería una forma muy mala de trabajar con una base de datos de una agenda. Para manejar datos se han definido cuatro operaciones básicas: Altas, bajas, modificaciones y consultas, además de la elaboración de listados y la compactación del fichero. En estos apartados veremos unas breves (muy breves) indicaciones de como conseguir que nuestros programas de QBasic puedan hacer esto con un fichero de acceso directo.

1.16.3.2 - ALTAS. AÑADIR REGISTROS

En nuestro ejemplo de la sección anterior veíamos como añadir registros a un archivo directo con la instrucción PUT. Lo hacíamos muy bien, solo que siempre empezábamos por la primera posición y de esta forma cada vez que ejecutamos el programa sobreescribíamos datos y perdíamos información.

La idea de todo esto es conseguir poder añadir información al fichero cada vez que ejecutemos el programa, siempre al final de lo que ya tenemos. Para conseguirlo no hay más que calcular cuantos registros tiene ya el fichero para meter el nuevo registro en la posición siguiente a la última.

Veamos como ejemplo un procedimiento SUB que pide al usuario el nombre y la edad de la persona y los añade al final del fichero. Suponemos que la estructura de TIPOPERSONA está definida en el programa principal y que el fichero también está ya abierto y tiene el descriptor #1.

```
'Procedimiento ALTA
'Añade un nuevo registro a agenda.dat
SUB Alta()
        DIM nuevaPersona AS TIPOPERSONA
        ultimoRegistro = LOF(#1) / LEN(NuevaPersona)
        INPUT "Nombre: ", nuevaPersona.nombre
        INPUT "Edad: ", nuevaPersona.edad
        PUT #1, ultimoRegistro + 1, nuevaPersona
END SUB
```

Si es la primera vez que llamamos al programa y el fichero está vacío no pasa nada, simplemente el nuevo registro se guarda en la posición 1. Si el archivo ya contiene datos, el nuevo registro se añadirá al final.

Otros algoritmos más elaborados permitirían aprovechar posibles "huecos" existentes en el archivo.

1.16.3.3 - CONSULTAS.
BÚSQUEDA SECUENCIAL DE REGISTROS

Una vez que tenemos información en el registro nos puede interesar buscar un nombre, por ejemplo, y que nos salga en la pantalla los otros datos de la persona, en nuestro ejemplo sólo la edad.

Para hacer esto hay complicados mecanismos de búsqueda muy rápida suponiendo que nuestro fichero esté ordenado. Como no es el caso, ya que la información está en el mismo orden en que la hemos ido metiendo, la única forma de encontrar algo es ir mirando registro por registro hasta que encontremos lo que buscamos o se acabe el fichero.

Vamos con un ejemplo que pregunta al usuario el nombre de la persona que quiere buscar y recorre el fichero hasta encontrarla o hasta que se acaba. Suponemos que ya está abierto el archivo y definido el tipo de datos registro, igual que en los ejemplos anteriores.

```
numregs = (LOF(1) / LEN(persona))
INPUT "Nombre a buscar: ", nombreBuscado$
n = 1
DO
       GET #1, n, persona
       n = n + 1
LOOP UNTIL RTRIM$(persona.nombre)=nombreBuscado$ OR n>numregs
IF n <= numregS THEN
       PRINT "Registro...:"; n-1
       PRINT "Nombre.....: "; persona.nombre
       PRINT "Edad.......:"; persona.edad
ELSE
       PRINT "No se ha encontrado"
END IF
```

Lo primero que hacemos es calcular cuantos registros tiene el fichero para ver asta donde podemos llegar. Después pedimos al usuario que escriba el nombre que quiere encontrar y lo guardamos en una variable del tipo adecuado, en este caso cadena.

En el siguiente bloque vamos recorriendo el fichero desde el primer registro usando un contador hasta que encontremos el nombre buscado o bien el contador llegue a superar el número de registros del fichero. Para hacer la comparación usamos la función RTRIM$ que lo que hace es quitar los espacios en blanco que hay a la derecha del nombre grabado en el registro porque como al definir el registro tenemos que usar cadenas de longitud fija los valores devueltos siempre contienen espacios al final para rellenar la cadena.

Cuando salimos del bucle lo que hacemos es comprobar el contador con un IF. Si se ha superado el límite es que no se ha encontrado, y en caso contrario es que se ha encontrado y podemos mostrar en pantalla todos los datos de la persona que siguen estando almacenados en la variable de registro, ya que después no se ha leído nada más. También incluimos el número del registro usando el contador menos uno, ya que antes de salir del bucle se sumó uno.

Si hay dos personas con el mismo nombre este código sólo encontrará la primera. Podemos hacer consultas por otros campos, o bien combinar varios campos con solo modificar la condición del bloque repetitivo que usamos para recorrer el archivo. Si queremos poder mostrar más de un registro encontrado habría que ir guardando los datos encontrados en un array del mismo tipo para después mostrarlos en una lista. Habría que limitar el número de registros mostrados al tamaño del array, igual que hace Windows 9x con las búsquedas de archivos y carpetas.

1.16.3.4 - MODIFICACIONES DE REGISTROS

Al trabajar con ficheros no nos podemos limitar a permitirle al usuario añadir datos, también hay que dejarle hacer modificaciones, ya que la información puede cambiar o el usuario se puede equivocar al escribir los datos y tendrá que rectificarlos. La forma de hacer una modificación consiste en sobreescribir un registro completo del fichero con la nueva información.

Una forma muy fácil de hacer esto es preguntar al usuario qué registro quiere modificar, mostrarle los datos a ver si de verdad son estos y a continuación pedirle la nueva información y grabarla en esa posición del fichero sobreescribiendo a la anterior.

Vamos a ver como sería (Archivos ya abiertos y tipos ya definidos):

```
numregS = (LOF(1) / LEN(persona))
INPUT "Número de registro a modificar: ", nummodif%
IF nummodif% < 1 OR nummodif% > numregS THEN
        PRINT "Número de registro incorrecto"
ELSE
        GET #1, nummodif%, persona
        PRINT "Nombre.....: "; persona.nombre
        PRINT "Edad.......:"; persona.edad
        PRINT "¿Es este el registro que quieres modificar? (S/N)"
        WHILE INKEY$ <> "": WEND 'Para limpiar INKEY$
        DO
                tecla$ = UCASE$(INKEY$)
        LOOP UNTIL tecla$ = "S" OR tecla$ = "N"
        IF tecla$ = "S" THEN
                PRINT "Escribe los nuevos datos..."
                INPUT "Nombre: ", persona.nombre
                INPUT "Edad: ", persona.edad
                PUT #1, nummodif%, persona
                PRINT "Los nuevos datos han sustituido a los
anteriores"
        END IF
END IF
```

Lo primero es pedir al usuario que escriba el número del registro, este número lo habrá averiguado en una consulta que haya hecho antes, por ejemplo. El número que escriba lo guardamos en una variable de tipo entero, ya que los registros nunca van a opder llevar decimales. A continuación comprobamos que el registro existe en el fichero, su número es mayor que 0 y no mayor que el

número máximo de registros que hemos calculado al principio. Sólo si es así seguimos, si no damos un mensaje de error y terminamos.

Si el registro existe lo mostramos en la pantalla y le preguntamos si es este el que quiere modificar. Si pulsa la S mayúscula le pedimos los nuevos datos y los grabamos en el fichero en la misma posición soreescribiendo los datos antiguos.

Al hacer modificaciones el tamaño del fichero no cambia.

El uso de la función INKEY$ para hacer preguntas tipo Sí o No al usuario lo veremos detalladamente en uno de los temas de ampliación del curso.

1.16.3.5 - BAJAS. BORRAR REGISTROS

En este modelo de programación de ficheros de acceso aleatorio no existe una forma específica de borrar información de un fichero, por esto lo que hacemos es simplemente sobreescribir el registro rellenándolo con espacios en blanco o valores cero, según corresponda, para que la información no aparezca en las consultas ni en los listados.

Vamos con un ejemplo de como hacerlo, casi igual al de las modificaciones.

```
numregS = (LOF(1) / LEN(persona))
INPUT "Número de registro a borrar: ", numborra%
IF numborra% < 1 OR numborra% > numregS THEN
        PRINT "Número de registro incorrecto"
ELSE
        GET #1, numborra%, persona
        PRINT "Nombre.....: "; persona.nombre
        PRINT "Edad.......:"; persona.edad
        PRINT "¿Es este el registro que quieres borrar? (S/N)"
        WHILE INKEY$ <> "": WEND 'Para limpiar INKEY$
        DO
                tecla$ = UCASE$(INKEY$)
        LOOP UNTIL tecla$ = "S" OR tecla$ = "N"
        IF tecla$ = "S" THEN
                persona.nombre = SPACE$(LEN(persona.nombre))
                persona.edad = 0
                PUT #1, numborra%, persona
                PRINT "Registro borrado"
        END IF
END IF
```

Como se puede ver, lo único que cambia es que en los mensajes en vez de decir "modificar" dice "borrar" y que sobreescribimos el registro con blancos y ceros en vez de con la información que escribe el usuario.

Lo único más extraño es que para asignar espacios en blanco al nombre en vez de hacer

```
persona.nombre = "                    "
```

que sería lo más fácil, usamos la función SPACE$ que nos devuelve una cadena de espacios en blanco, y para calcular cuantos espacios usamos la función LEN que nos devuelve el tamaño del dato miembro. De esta forma no nos tenemos que acordar de cuanto largo era ese dato y si modificamos este tamaño durante el desarrollo del programa no tenemos que modificar esta instrucción, evitando posibles errores.

Visto esto se puede comprender que el tamaño de los ficheros directos nunca se va a reducir, ni aunque borremos registros. Para solucionar este problema lo que haremos será compactar el fichero, en el apartado siguiente.

1.16.3.6 - COMPACTACIÓN DE FICHEROS DIRECTOS

Como hemos visto, al borrar registros el tamaño del fichero nunca se reduce, y en nuestro modelo de programación sencillo los nuevos registros siempre se añaden al final, con lo que nuestro fichero va a crecer y crecer siempre pudiendo contener muchos registros vacíos que lo único que hacen es ocupar sitio y hacer que todo vaya un poco más lento.

Para solucionar este problema en las bases de datos se ha definido una tarea de mantenimiento que consiste en "compactar" los ficheros. No tiene nada que ver con lo que hacen programas como WinZip, esto es "comprimir" ficheros y la forma de programar estas operaciones es terriblemente más complicada.

Imaginemos que en la vida real tenemos una agenda de papel llena de datos de personas de los cuales unos sirven y otros están tachados porque son incorrectos o se han quedado obsoletos. Además de llevar siempre un tocho de hojas muy gordo, a la hora de buscar algo tendremos que pasar siempre muchas hojas, perdiendo más tiempo del necesario. Una forma muy cómoda de solucionar este problema sería conseguir una agenda nueva para quedarnos sólo con los datos que queremos conservar. Para hacer esto recorreríamos la agenda vieja y solo iríamos copiando en la nueva los datos que no están tachados. Al final tiramos la vieja y nos quedamos con la nueva para seguir usándola a partir de ahora.

Este trozo de código hace justamente eso mismo con nuestro fichero informático de la base de datos. Suponemos que el fichero de nuestra agenda ya esta abierto y los tipos de datos definidos.

```
numregs = (LOF(1) / LEN(registro))
OPEN "Agenda.tmp" FOR RANDOM AS #2 LEN = LEN(registro)
n1 = 1
n2 = 1
WHILE n1 <= numregS
       GET #1, n1, registro
       IF registro.nombre <> SPACE$(LEN(registro.nombre)) OR
registro.edad <> 0 THEN
              PUT #2, n2, registro
              n2 = n2 + 1
```

```
        END IF
        n1 = n1 + 1
WEND

CLOSE

KILL "Agenda.dat"
NAME "Agenda.tmp" AS "Agenda.dat"

OPEN "Agenda.dat" FOR RANDOM AS #1 LEN = LEN(registro)
'Seguimos con el programa...
```

Vamos a ver como funciona esto:

- Lo primero que hacemos, como siempre, es calcular el número de registros que tiene el fichero para saber hasta donde podemos llegar.
- A continuación abrimos un fichero llamado "Agenda.tmp" (La extensión .tmp significa "temporal") y le asignamos el descriptor #2. Este fichero debe ser abierto con la misma estructura que el otro y suponemos que no existe y se crea uno nuevo y vacío al abrirlo.
- Antes de entrar en el bucle inicializamos dos contadores al valor 1, uno nos va a servir para recorrer el fichero viejo y otro para recorrer el nuevo. Empezamos por 1 que es el primer registro.
- Ahora usamos un bucle "Mientras" para recorrer el fichero viejo desde el principio hasta el final. Dentro de este bucle leemos cada registro y si no está completamente vacío lo copiamos al nuevo fichero e incrementamos ambos contadores. Si está vacío no hacemos nada y solo incrementamos el primer contador para poder seguir recorriendo el fichero viejo.
- Para comprobar si el registro está completamente vacío habrá que usar una expresión condicional que puede llegar a ser muy larga porque hay que comprobar cada dato miembro. En nuestro ejemplo esta expresión aparece ocupando dos líneas, pero si la escribes en QBasic tendrá que ser toda seguida en la misma linea, aunque sea muy larga. QBasic soporta líneas de hasta 256 caracteres de larga, si sigue sin caber tendrás que cambiar los nombres de las variables por otros más cortos, aunque en esta ocasión sean menos descriptivos.
- Una vez que salimos del bucle tendremos el fichero "Agenda.dat" tal como estaba y el nuevo "Agenda.tmp" compactado solo con los datos que nos interesan.
- Para proceder al intercambio de los nombres de los ficheros y quedrnos solo con el nuevo podemos seguir estos pasos
    - Cerrar los dos ficheros, usando la orden CLOSE sin descriptor.
    - Borrar "Agenda.dat" usando la orden KILL, que veremos con detalle más adelante.
    - Cambiarle el nombre a "Agenda.tmp" por "Agenda.dat" usando la orden NAME, que también veremos en próximos temas.
    - Volver a abrir "Agenda.dat" con el descriptor #1 para seguir trabajando con él normalmente desde el resto del programa.

Esta operación de mantenimiento la tendrá que ejecutar el usuario de vez en cuando, especialmente después de haber borrado muchos registros. En programas mucho más sofisticados se podría hacer que se ejecute cada x días o tras haber borrado un determinado número de registros.

1.16.3.7 - ELABORACIÓN DE LISTADOS DE DATOS

Los listados son una parte muy útil de todo programa que trabaje con datos, para que el usuario pueda ver toda la información a la vez y no registro por registro. En los programas modernos se suele hablar de "informes" para referirse a listados con un determinado formato que presentan los registros que cumplen cierta condición.

Aquí nos limitaremos a escribir en la pantalla un listado encolumnado de todos los registros que haya en nuestro fichero. La forma de dibujar un listado en la pantalla es bastante sencilla en principio. Bastaría con recorrer el fichero y escribir cada registro en una línea de la pantalla, pero surgen algunos problemas que vamos a ir solucionando más adelante.

Vamos con un ejemplo, de lo más cutre, que recorre el fichero y escribe su contenido línea por línea todo seguido. Suponemos que ya tenemos abierto nuestro fichero y definido el tipo registro, igual que en los ejemplos anteriores.

```
CLS
numregs=LOF(1) - LEN(persona)
c=1
WHILE c <= numregs
       GET #1, c, persona
       PRINT c; persona.nombre; persona.edad
       c=c+1
WEND
```

El resultado por pantalla podría ser:

```
1Pepe                25
2                    0
3                    0
4Manolo García García32
5Paco                19
6                    0
7                    0
8                    0
9Ana                 40
10Juan Carlos         28
11Fernando            21
```

Lo primero, como siempre, calcular cuantos registros tiene el fichero y después irlos leyendo y sacando por pantalla desde dentro de un bloque MIENTRAS. Se podía haber hecho con un PARA, pero como después vamos a tener que hacerlo con un MIENTRAS de todas formas, mejor empezar así.

El listado aparece más o menos encolumnado porque las variables de cadena (nombre) se devuelven con espacios en blanco detrás hasta completar su longitud total, pero los enteros no ocuparían siempre lo mismo y nos estropearían el encolumnado, cosa que pasaría a partir del registro 10 donde el nombre sería empujado un lugar hacia la derecha. Más adelante iremos solucionando esto.

Como se puede ver en el resultado del ejemplo anterior, han aparecido también los registros borrados (Sin nombre y con edad=0). Para evitar esto lo que hacemos comprobar con una condición justo antes de escribir cada línea que el registro no ha sido borrado. Sería así:

```
CLS
numregs=LOF(1) - LEN(persona)
c=1
WHILE c <= numregs
       GET #1, c, persona
       IF persona.nombre<>SPACE$(LEN(persona.nombre)) OR
persona.edad>0 THEN
              PRINT c; persona.nombre; persona.edad
       END IF
       c=c+1
WEND
```

Y el resultado podría ser:

```
1Pepe                25
4Manolo García García32
5Paco                19
9Ana                 40
10Juan Carlos         28
11Fernando            21
```

Vemos que ya no aparecen los registros que fueron borrados. La condición que hacemos es que el nombre (texto) no sea todo espacios o que la edad sea mayor que cero. Ahora sólo aparecen los registros que nos interesan, pero todavía se ve muy feo vamos a dibujarle un encabezado a la lista y dejar un poco de espacio entre las columnas para que no quede tan junto.

```
CLS
numregs=LOF(1) - LEN(persona)
c=1
PRINT "Registro     Nombre               Edad"
PRINT "============ ==================== ===="
WHILE c <= numregs
       GET #1, c, persona
       IF persona.nombre<>SPACE$(LEN(persona.nombre)) OR
persona.edad>0 THEN
              PRINT c, persona.nombre; " "; persona.edad
       END IF
       c=c+1
WEND
```

Y el resultado, más bonito sería:

```
Registro     Nombre               Edad
============ ==================== ====
1            Pepe                 25
4            Manolo García García 32
5            Paco                 19
9            Ana                  40
10           Juan Carlos          28
11           Fernando             21
```

Dibujamos el encabezado una vez al principio. La línea con guiones o signos igual nos puede servir para orientarnos con los tamaños de los campos. Dentro del bucle en la instrucción PRINT si separamos las variables por comas se pasa a la siguiente posición de tabulación y nos evitamos el problema del diferente número de cifras, aunque se desperdicia mucho espacio porque las posiciones de tabulación an QBasic miden 13 caracteres. Podemos meter espacios entre comillas para separar los campos. En este caso como hay muy pocos campos y hemos podido dejar los números enteros para el final ha salido medio bien, pero en casos más complicados sería necesario usar la instrucción TAB para definir las columnas. Veamos como se haría.

```
CLS
numregs=LOF(1) - LEN(persona)
c=1
PRINT "Reg"; TAB(5); "Nombre"; TAB(26); "Edad"
PRINT "==="; TAB(5); "=================="; TAB(26); "===="
WHILE c <= numregs
       GET #1, c, persona
       IF persona.nombre<>SPACE$(LEN(persona.nombre)) OR
persona.edad>0 THEN
              PRINT c; TAB(5); persona.nombre; TAB(26);
persona.edad
       END IF
       c=c+1
WEND
```

Y el resultado, todavía mejor.

```
Reg Nombre               Edad
=== ================== ====
1   Pepe                 25
4   Manolo García García 32
5   Paco                 19
9   Ana                  40
10  Juan Carlos          28
11  Fernando             21
```

El parámetro que le pasamos a la orden TAB debe ser un entero que indica la posición a la que saltará el cursor dentro de la línea actual. Recordemos que cada línea tiene 80 caracteres de ancho y que la primera posición por la izquierda es la 1. La orden TAB se utiliza como argumento de la orden PRINT, igual que si fuera una función, solo que en vez de devolver un valor lo que hace es llevar al cursor hasta la posición especificada. En el encabezado nos podíamos ahorrar las ordenes TAB y poner directamente los espacios, porque como todo es constante no va a variar nada. Para saber las posiciones lo más cómodo es ir probando hasta que cada cosa caiga en su sitio, debajo de su título.

Ya se empieza a parecer a un listado. Siempre habrá que ajustar las columnas según nuestras necesidades y si no caben de ancho la única solución posible con lo que sabemos hasta ahora es no escribir todos los campos.

Lo último que vamos a solucionar es el terrible problema de que si tenemos muchos registros el listado avanzará rápidamente y solo veremos los últimos veintitantos registros que caben en la pantalla. Pasa lo mismo que con la orden

DIR de MS-DOS si no ponemos el modificador /P. Para conseguir esto lo que hacemos es usar un segundo contador que vamos incrementando en cada línea y cuanto veamos que llega a 20 lo ponemos a cero, y hacemos una pausa, cuando el usuario pulse una tecla borramos la pantalla, dibujamos el encabezado de nuevo y seguimos. Vamos con el ejemplo:

```
CLS
numregs=LOF(1) - LEN(persona)
c=1
c2=1
PRINT "Reg Nombre                Edad"
PRINT "=== ==================== ====="
WHILE c <= numregs
       GET #1, c, persona
       IF persona.nombre<>SPACE$(LEN(persona.nombre)) OR
persona.edad>0 THEN
              PRINT c; TAB(5); persona.nombre; TAB(26);
persona.edad
       END IF
       c=c+1
       c2=c2+1
       IF c2>20 THEN
              c2=1
              PRINT "Pulsa cualquier tecla para seguir..."
              WHILE INKEY$<>"":WEND
              DO:LOOP WHILE INKEY$=""
              CLS
              PRINT "Reg Nombre                Edad"
              PRINT "=== ==================== ====="
       END IF
WEND
```

El resultado, si hay menos de 20 registros sería igual que antes, y si hay más sería que se van viendo de 20 en 20.

Este ejemplo se puede mejorar para que las instrucciones donde se dibuja el encabezado sólo aparezcan una vez, pero aquí no nos vamos a complicar tanto. De todas formas en el tema de manejo de la pantalla de texto que hay en la sección de temas de ampliación de este curso aparen las instrucciones a usar para hacer listados desplazables parecidos a los de Windows que son mucho más "bonitos" y no tienen estos problemas.

1.16.4 - BREVE IDEA SOBRE BASES DE DATOS

En informática se conoce como base de datos a un mecanismo capaz de almacenar información para después poder recuperarla y manejarla con facilidad.

Visto así, los archivos de ejemplo de agendas que hemos creado en los apartados anteriores son bases de datos, pero cuando hablamos de bases de datos pensamos en programas que manejan gran cantidad de información de forma muy estructurada y que permiten realizar consultas muy complicadas.

Al hablar de bases de datos hay que distinguir dos cosas muy importantes. Por un lado tenemos la base de datos en sí, que es la información grabada en los discos, y por otro lado tenemos el Sistema de Gestión de Bases de Datos (SGBD o DBMS en inglés) que es el programa encargado de manejar toda la información. Es lo que sería un programa como MS Access o MySQL.

Las bases de datos que manejan estos programas casi nunca contienen una sola tabla, es decir, un solo archivo de datos, sino que contienen muchas tablas relacionadas entre si. Por eso se habla de bases de datos relacionales. El hecho de que las tablas estén relacionadas unas con otras implica que al hacer un cambio en una de ellas, posiblemente se vean afectadas otras para que la información siga teniendo consistencia. También estos sistemas de bases de datos usan muchos mecanismos como índices o algoritmos de ordenación para conseguir que el acceso a la información durante las consultas sea un proceso mucho más rápido y completamente fiable.

Nosotros, en nuestros programas estamos integrando una pequeñísima parte de los mecanismos que usan los SGBD para poder manejar la información de nuestros archivos, pero conforme nuestras necesidades aumenten y ya usemos otros lenguajes de programación, será mucho más sencillo usar los servicios de un auténtico SGBD para manejar nuestra información, en vez de programarlo nosotros. Esto quiere decir que el uso de lo que hemos visto en este tema de archivos de acceso directo se limitará a el mantenimiento de pequeños y sencillos conjuntos de datos cuando sea más comodo manejarlos de forma directa que de forma secuencial, pero desde el momento en que intervienen varios archivos de datos necesitaremos usar las funciones de uno de estos potentes SGBDs integrados en los modernos lenguajes de programación.

1.16.5 - OTRAS INSTRUCCIONES RELACIONADAS CON FICHEROS

Ya hemos visto como crear ficheros y leer o modificar la información almacenada en los mismos usando diferentes métodos de programación. Para terminar el tema de ficheros es necesario conocer algunas instrucciones que tiene QBasic para trabajar con ficheros enteros que haya en el sistema de archivos. Estas instrucciones nos ayudarán a trabajar con los ficheros sin tener que recurrir a ejecutar ordenes del sistema operativo.

Vamos con ellas.

`CHDIR ruta$`

Cambia el directorio activo. El directorio activo es el que usará QBasic para buscar los archivos que tiene que abrir o para crearlos si no se especifica una ruta en la instrucción OPEN. Puede ser difícil determinar el directorio activo, por lo que conviene especificar uno con esta orden antes de abrir o crear ficheros.

Si ejecutamos QBasic desde un icono de acceso directo de Windows el directorio activo será el que aparece en el cuadro "Carpeta de trabajo" de la sección "Programa" de las propiedades del acceso directo.

Si ejecutamos QBasic desde MS-DOS el directorio activo será donde nos encontrábamos cuando entramos en QBasic.

La ruta que hay que especificar en esta instrucción puede ser un literal entre comillas o una variable de cadena que contenga la ruta adecuada. Si el directorio no existe se producirá un error de tiempo de ejecución.

Hay que recordar que estamos en MS-DOS y por lo tanto solo podemos trabajar con nombres de directorios y archivos de no más de 8 caracteres y sin espacios, por lo tanto los directorios "Archivos de Programas" y "Mis documentos" tendrán nombres parecidos a "ARCHIV~1" y "MISDOC~1". Para asegurarse de cuales son estos nombres hay que mirar la casilla "Nombre MS-DOS" de la ventana de propiedades del directorio correspondiente en Windows.

Para escribir el carácter "Gurruño" (~) hay que pulsar la tecla "Alternativa" y simultáneamente escribir con el teclado numérico la cifra 126, correspondiente a su código ASCII.

La siguiente orden nos permite crear un directorio:

```
MKDIR ruta$
```

El directorio que creemos ya estará disponible en el sistema de archivos para usarlo desde nuestro programa o desde cualquier otro. Si la ruta que especificamos ya existe se producirá un error de tiempo de ejecución.

Es importante comprender que al crear un directorio este no pasa a ser el directorio activo. Para hacerlo habrá que usar una instrucción CHDIR.

Ahora vamos a ver como borrar un directorio existente:

```
RMDIR ruta$
```

Esta instrucción borra el directorio que le pasemos como parámetro. Si no existe se producirá un error. También se producirá el mismo error si el directorio no está vacío, es decir, contiene algún archivo o más subdirectorios. Habrá que borrarlos antes como veremos más adelante. Normalmente debemos tener mucho cuidado con lo que borramos desde nuestros programas para no hacer ningún estropicio con la información que tenemos guardada en los discos.

Podemos ver los ficheros y subdirectorios de un directorio con la siguiente instrucción:

```
FILES
```

Esta instrucción trabaja de forma parecida a la orden "DIR /W" de MS-DOS. Nos dibujará un listado a cuatro columnas con el contenido de un directorio, encabezada por el nombre del propio directorio y terminada por el espacio en bytes disponible en el disco. Los subdirectorios se identifican porque llevan a la derecha el rótulo "< DIR >".

Esta instrucción nos puede resultar algo útil en los programas que hemos hecho hasta ahora ya que la información se escribe en pantalla una línea debajo de

otra, pero cuando hagamos diseños de pantallas más elaborados ya no será recomendable usar esta instrucción porque el listado de directorio se saltará todas nuestras especificaciones de pantalla, incluso ahora si el listado es muy largo sólo se ven las últimas líneas.

Si usamos la orden FILES sin parámetros se mostrarán todos los ficheros y subdirectorios del directorio activo, pero podemos especificar rutas, nombres de archivos y caracterescomodín (? y *) para adecuar el listado a nuestras necesidades. Veamos unos cuantos ejemplos:

```
FILES "*.txt"
```

Lista todos los ficheros con extensión TXT que haya en el directorio activo.

```
FILES "JUEGO.*"
```

Lista todos los ficheros que se llamen JUEGO, sea cual sea su extensión, del directorio activo.

```
FILES "C:\BASIC\PROGS\*.*"
```

Lista todos lo que haya en el directorio BASIC\PROGS de la unidad C:.

```
FILES "Imagen?.BMP"
```

Lista todos los archivos cuyo nombre empiece por "Imagen" y vaya seguido por exactamente un carácter, su extensión sea BMP y stén en el directorio activo. Aparecerían por ejemplo los archivos Imagen2.BMP o Imagen7.BMP, pero nunca el archivo Imagen14.BMP ya que contiene dos números.

Si la orden FILES no encuentra nada que listar se produce un error de tiempo de ejecución. Si listamos un directorio vacío no se producirá ningún error.

Ya hemos visto como borrar directorios, ahora vamos a ver como borrar archivos:

```
KILL Archivo$
```

Esta instrucción borra archivos, podemos usar una ruta y/o los comodines * y ?, cosa que puede ser peligrosa porque se borrarían varios ficheros a la vez, una vez más recomendar prudencia al usar esta orden para evitar estropicios irreparables, ya que en MS-DOS normalmente no tenemos papelera de reciclaje.

Si no especificamos una ruta se entenderá que estamos trabajando sobre el directorio activo. Si el archivo que queremos borrar no existe se producirá un error de tiempo de ejecución.

No confundir esta instrucción con la orden KILL de Linux que detiene procesos, es decir, obliga a terminar la ejecución de un programa pero no borra nada del disco.

Por último vamos a ver como cambiar el nombre un fichero:

```
NAME viejo$ AS nuevo$
```

viejo$ es el nombre actual del archivo que queremos renombrar, que debe existir para que no se produzcan errores.

nuevo$ tiene que ser el nuevo nombre que le vamos a dar al archivo, y que no debe existir todavía.

Por ejemplo, para cambiar el nombre al archivo base.dat del directorio actual por base.bak tendríamos que usar la instrucción

```
NAME base.dat AS base.bak
```

En el nombre del archivo ya existente (la primera parte antes de AS) podemos especificar una ruta de directorios.

Para copiar o mover archivos no tenemos una instrucción específica en QBasic. Tendremos que crearnos un modulo que lo abra en modo binario y lo copie carácter por carácter en un archivo nuevo en la otra ubicación, si lo que queríamos era moverlo después habrá que borrar el original. Para hacer esto y otras cosas más complicadas puede ser más cómodo llamar a la orden correspondiente de MS-DOS como se verá en temas posteriores.

TEMA 1.17
CONTROL DE ERRORES

- **1.17.1 - Instrucción GOTO. La más conocida y odiada de BASIC**
- **1.17.2 - Introducción a los errores**
- **1.17.3 - Control de errores**

1.17.1 - INSTRUCCIÓN GOTO.
LA MÁS CONOCIDA Y ODIADA DE BASIC

Empecemos con un ejemplo "muy malo":

```
       CLS
Pricipio:
       PRINT "Esto es el principio"
       GOTO Final
Enmedio:
       PRINT "Esto es lo de en medio, pero no lo verás"
Final:
       PRINT "Esto es el Final"
```

Que daría como resultado en pantalla:

```
Esto es el principio
Esto es el final
```

Ahora veamos como funciona:

- Borra la pantalla.
- Pasa por una etiqueta definida como Principio.
- Escribe "Esto es el principio"
- Busca una etiqueta que se llame "Final" y cuando la encuentra salta y sigue a partir de allí.
- Escribe "Esto es el final".
- Como no hay nada más, el programa termina.

En este programa hemos definido tres etiquetas. Para definir etiquetas usamos las mismas normas que para los nombres de variables y al final le colocamos el carácter dos puntos (:). Las etiquetas siempre van al principio de la línea y si usamos etiquetas es útil encolumnar todas las demás instrucciones más a la derecha, para verlas al vuelo.

Podemos definir etiquetas en cualquier módulo de nuestro programa, pero QBasic solo las "encontrará" si están definidas en el módulo principal. Por esto no tiene sentido definir etiquetas dentro de procedimientos y funciones.

Las etiquetas no hacen nada, simplemente están ahí. Únicamente nos sirven para poder "Saltar" a ellas usando instrucciones especiales como la famosa GOTO (Go to..., ir a...) que salta incondicional e inmediatamente a la etiqueta que le indiquemos para seguir desde allí con la ejecución del programa.

Sencillo ¿No?. En verdad es muy sencillo pero puede dar lugar a toda una variedad de problemas lógicos que pueden ser casi imposibles de detectar, aislar y solucionar. Imagina un programa donde en vez de tres etiquetas haya trescientas y entre ellas haya centenares de instrucciones GOTO que dirigen la ejecución del programa a un sitio lejano de nuestro listado. Una mala planificación del programa seguramente hará que bloques enteros no se lleguen a ejecutar nunca o que se produzcan bucles infinitos que lleguen a bloquear el programa. Por eso a los programas que usan estas instrucciones se les conoce como "Código Espagueti".

En los lenguajes de bajo nivel como es el caso de los ensambladores y en lenguajes de guiones muy sencillos como el que se utiliza para construir los ficheros .BAT de MS-DOS se usan intensivamente las etiquetas y las instrucciones de bifurcación o salto incondicional, pero los modernos lenguajes de programación estructurada (QBasic lo es) nos dan la posibilidad de usar estructuras de control iterativas (Bucles PARA, MIENTRAS y REPETIR), así como los procedimientos y las funciones que nos permiten en todo caso saber como funciona nuestro programa para poder solucionar posibles errores. Por esto es muy recomendable **EVITAR A TODA COSTA EL USO DE ESTA METODOLOGÍA DE PROGRAMACIÓN CON INSTRUCCIONES GOTO Y SIMILARES** en los lenguajes de alto nivel, salvo en el caso de las instrucciones de manejo de errores que vamos a ver en los siguientes apartados y que son el motivo de esta explicación.

1.17.2 - INTRODUCCIÓN A LOS ERRORES

Hemos ido viendo a lo largo de los temas anteriores que hay tres tipos de errores:

- Errores de compilación
- Errores de tiempo de ejecución
- Errores lógicos

Los primeros suelen ser cosas mal escritas o que no cumplen con las normas de sintaxis del lenguaje de programación. La mayoría de ellos en QBasic serán mostrados conforme vamos escribiendo nuestro código al pasar de línea, o en todo caso al intentar ejecutar el programa. Tendremos que corregirlos modificando lo que este mal para poder iniciar el programa.

Los errores de tiempo de ejecución se producen durante la ejecución del programa y son provocados por intentar hacer algo que no está permitido, como dividir entre cero, o bien porque alguno de los dispositivos del ordenador falle (Memoria agotada, no se encuentra un fichero, la impresora no tiene papel, etc...). Estos errores hacen que el programa se detenga. En QBasic se volverá al editor y se mostrará un recuadro con el mensaje apropiado. En otros

lenguajes o en programas ya compilados puede salir desde un simple mensaje hasta una pantalla azul típica de Windows o lo más normal es que el ordenador deje de responder y haya que reiniciarlo.

Los errores lógicos se deben a una mala planificación de los algoritmos, lo que da lugar, en el mejor de los casos, a que el programa funcione correctamente pero no haga lo que queremos y no resuelva el problema para el que ha sido diseñado, o lo más normal es que llegue a provocar un error de tiempo de ejecución porque se llene el espacio disponible en memoria o se entre en un bucle infinito.

En este tema vamos a ver cómo conseguir que cuando se produzca un error de tiempo de ejecución el programa no quede detenido inmediatamente, sino que se salga de él de una forma un poco más "elegante" o incluso se pueda solucionar el problema y seguir normalmente con la ejecución del programa.

Usando las técnicas de programación estructurada que hemos visto en este curso hasta antes del tema de ficheros si hacemos los programas bien NO TIENEN PORQUÉ PRODUCIRSE ERRORES DE TIEMPO DE EJECUCIÓN. Nosotros somos los responsables de evitar que los bucles sean infinitos, de no hacer referencias a índices de arrays que no existen, de que nada se llegue a dividir entre cero, etc... La única fuente potencial de errores son los datos que pedimos al usuario, pero si depuramos los datos de entrada de forma que no se acepten valores no permitidos como vimos en el tema de bucles podemos asegurar en la amplia mayoría de los casos que nuestro programa va a funcionar bien siempre, ya que solo acepta datos válidos.

El problema llega cuando empezamos a trabajar con dispositivos externos como son las unidades de disco donde se guardan los ficheros. Por muy bien hecho que esté nuestro programa no podemos evitar que el usuario quiera abrir un fichero que no existe, o de un disquete que todavía no ha introducido en la unidad correspondiente, o que se ha estropeado, que quiera escribir en un CD-ROM, en un disco que no tiene espacio suficiente o que está protegido contra escrituea. Sólo en este caso de los ficheros y en otros también especiales como cuando veamos cómo usar las impresoras será necesario y recomendable programar rutinas de manejo de errores de las que vamos a ver en el siguiente apartado. En otros casos no nos merece la pena usar esto ya que las instrucciones GOTO ofrecen una forma especialmente mala de estructurar los programas y podemos despreciar la posibilidad de que se produzca algún error raro como que se agote la memoria por culpa de otro programa y el nuestro no pueda seguir funcionando.

1.17.3 - CONTROL DE ERRORES

Para activar el control de errores en QBasic usaremos la siguiente instrucción:

```
ON ERROR GOTO linea
```

Dónde linea es el nombre de una etiqueta que se haya definido en el programa principal.

Esto quiere decir "Cuando se produzca un error ve inmediatamente a la etiqueta que se llama linea y sigue desde allí".
De esta forma conseguimos que el programa no se detenga inmediatamente al producirse un error.

Como hemos dicho antes, solo debemos controlar los errores en instrucciones peligrosas como es el caso de las que abren los ficheros (OPEN). Una ver pasadas estas instrucciones debemos desactivar el manejo de errores usando la instrucción:

```
ON ERROR GOTO 0
```

Lo último de esta instrucción es un cero. A partir de ahora cuando se produzca un error ya no habrá manejo de errores y el programa se detendrá, pero si lo hemos hecho bien no tienen porqué producirse errores.

Lo normal es activar el manejo de errores al principio de un procedimiento que, por ejemplo, abra un fichero y desactivarlo al final, antes de salir. De esta forma tenemos claramente delimitado donde usamos el manejo de errores, que estará activo solo mientras se ejecute ese módulo, y no en el resto del programa que no usa instrucciones "conflictivas".

Vamos con un ejemplo completo:

```
'Programa principal
CLS
INPUT "Nombre del fichero: ", fich$
MuestraTamanno fich$

END

manejaerrores:

SELECT CASE ERR
   CASE 53
      PRINT "No se ha encontrado el archivo"
   CASE 68
      PRINT "No se ha encontrado la unidad de disco"
   CASE 71
      PRINT "No se ha encontrado el disco en la unidad"
   CASE 76
      PRINT "No se ha encontrado el directorio"
   CASE ELSE
      PRINT "Se ha producido el error"; ERR
END SELECT

END

'Este procedimiento abre el archivo, muestra el tamaño y cierra
'Entrada: f$: El nombre del archivo
SUB MuestraTamanno (f$)
   ON ERROR GOTO manejaerrores
   OPEN f$ FOR INPUT AS #1
   PRINT "El fichero ocupa"; LOF(1); "bytes."
   CLOSE
   ON ERROR GOTO 0
```

```
END SUB
```

Este programa pide al usuario que escriba el nombre de un fichero para abrirlo y mostrar su tamaño usando la función LOF que ya conocemos.

Si escribimos un nombre de archivo válido no se activa para nada el manejo de errores y el programa funciona normalmente, nos calcula el tamaño en bytes del archivo y termina al llegar a la instrucción END que hay justo después de la llamada al procedimiento. El resultado podría ser:

```
Nombre del fichero: c:\autoexec.bat
El fichero ocupa 219 bytes.
```

Veamos como funciona el programa paso a paso en caso de que el fichero exista y no se produzca el error:

- Entramos al módulo principal.
- Borramos la pantalla con CLS.
- Pedimos al usuario que escriba el nombre de un fichero y lo guardamos en la variable fich$:
- Entramos al procedimiento Muestratamanno y le pasamos la variable fich$ que contiene el nombre que ha escrito el usuario.
- Ya dentro del procedimiento, activamos el control de errores.
- Abrimos el fichero, que existe, para lectura y le asignamos el descriptor #1
- Calculamos el tamaño del fichero abierto #1 con la función LOF y lo escribimos en pantalla junto con un mensaje.
- Cerramos todos los ficheros abiertos (El único que hay).
- Desactivamos el control de errores y salimos del módulo porque no hay más instrucciones.
- De vuelta al programa principal la siguiente instrucción que encontramos es END que hace que el programa termine, la usamos para que no se ejecuten las siguientes instrucciones que son las del control de errores.

Aquí encontramos como novedad la instrucción END que hace que el programa termine de forma normal aunque no hayamos llegado al final del listado. La tendremos que poner siempre en el módulo principal antes de la etiqueta del manejador de errores para evitar que estas instrucciones se ejecuten en caso normal de que no haya un error. En otros casos no es recomendable usar esta instrucción, los programas y módulos deben empezar por el principio y terminar por el final del listado.

Ahora vamos a ver como funcionaría el programa paso a paso en el caso de que quisiéramos abrir un fichero que no existe y se produjera el error.

- Entramos al módulo principal.
- Borramos la pantalla con CLS.
- Pedimos al usuario que escriba el nombre de un fichero y lo guardamos en la variable fich$:
- Entramos al procedimiento Muestratamanno y le pasamos la variable fich$ que contiene el nombre que ha escrito el usuario.
- Ya dentro del procedimiento, activamos el control de errores.

- Intentamos abrir el fichero como no existe se produciría un error de tiempo de ejecución que es detectado por el control de errores que continúa inmediatamente con la ejecución del programa a partir de la línea definida como manejaerrores en el programa principal.
- Después de esta línea encontramos un SELECT CASE que usa la función ERR. Esta función devuelve el número del error que se ha producido. Cada error tiene un número único que nos sirve para identificarlo y hacer una cosa u otra según lo que haya ocurrido. En este caso lo usaremos para sacar un mensaje adecuado diciendo al usuario lo que ha pasado.
- Después del SELECT CASE, el programa termina, pero lo hace normalmente porque se ha acabado el listado, y no porque se ha producido un error, cosa que hay que evitar a toda costa.

Hay que tener claro que al producirse el error el fichero no se ha abierto, y por lo tanto el descriptor #1 no contiene referencia a ningún fichero, por lo que no podemos intentar hacer nada con él, ni siquiera cerrar el fichero.

Hay un gran número de códigos de error, puedes ver una lista completa de ellos en la ayuda en pantalla de Qbasic. Nunca es necesario controlarlos todos, por ejemplo si estamos trabajando con ficheros solo pondremos algunos de los relacionados con ficheros. Siempre es conveniente usar algo como CASE ELSE para que quede controlado alguno que se nos haya podido escapar, en este caso bastará con mostrar el número por pantalla para poder identificar el error de alguna forma, pero siempre haciendo que el programa termine bien.

Se pueden hacer manejos de errores mucho más elaborados que lleguen a solucionar el problema y el programa pueda seguir funcionando como si nada. Imagina que queremos abrir un fichero .INI dónde está la configuración del programa, y este se ha borrado. En este caso de forma totalmente transparente al usuario podríamos crear uno nuevo vacío o con valores por defecto y dejarlo abierto para que el programa siga trabajando con él normalmente. En este caso usaríamos la instrucción RESUME para continuar con el programa por la línea siguiente a la que produjo el error, una vez que ya hayamos abierto el nuevo fichero para escritura, hayamos escrito en él lo que sea, lo hayamos cerrado y lo hayamos vuelto a abrir para lectura, todo esto en el manejador de errores.

TEMA 1.18
LLAMADA A PROGRAMAS EXTERNOS

Lo último que nos queda por ver en la sección de temas básicos del curso de introducción a la programación es cómo conseguir que nuestro programa sea capaz de iniciar otros programas externos, ya sean ejecutables .EXE u ordenes de MS-DOS internas o externas o bien otros programas de QBasic.

Para conseguir lo primero usaremos la instrucción SHELL seguida de la orden de MS-DOS o del programa ejecutable o programa BAT que queremos ejecutar. Vamos con unos cuantos ejemplos:

```
SHELL "dir"
```

Hace que en la pantalla de nuestro programa aparezca el listado del contenido del directorio activo.

```
SHELL "dir c:\basic\juegos"
```

Hace que en la pantalla de nuestro programa aparezca el listado del contenido del directorio c:\basic\juegos. Si este directorio no existe no se producirá ningún error en nuestro programa, simplemente la orden DIR nos sacará un mensaje diciendo que no se ha encontrado el achivo.

```
SHELL "lote.bat"
```

Ejecuta el archivo de proceso por lotes llamado lote.bat. Lo buscará en el directorio activo y si no lo encuentra lo buscará en los directorios que aparecen en la variable de entorno PATH del MS-DOS. Si no lo encuentra sacará el típico mensaje "Comando o nombre de archivo incorrecto", pero en nuestro programa no ocurrirá ningún error.

```
SHELL "winmine"
```

En la mayoría de los casos, iniciará el buscaminas de Windows, un programa ejecutable. Si no lo encuentra pasará lo mismo que en el ejemplo anterior, pero tampoco generará ningún error.

```
SHELL
```

Si no escribimos ningún argumento, se abrirá una instancia de MS-DOS dentro de nuestro programa de QBasic. Aquí el usuario podrá ejecutar órdenes como se hace normalmente. Para volver al programa tendrá que escribir EXIT y pulsar Enter. Habrá que advertir esto al usuario antes de entrar para que no se quede "encerrado" si no sabe como salir. También es peligroso ejecutar determinados programas ya que en estos entornos los recursos de memoria son muy limitados.

Esta instrucción SHELL nos puede servir para ejecutar ordenes de MS-DOS o pequeños programas que nos pueden hacer falta usar.
Si ejecutamos algo de MS-DOS, que no es un sistema multitarea, nuestro programa quedará detenido hasta que este programa termine. En el caso que

llamemos a algún programa Windows, como el caso del buscaminas, nuestro programa lo iniciará y seguirá funcionando.

Normalmente no será necesario llamar a programas externos, salvo en situaciones muy específicas, y siempre deberán ser programas sencillos que consuman muy pocos recursos, si llamamos a algún programa muy grande como Windows desde QBasic y fallan los sistemas de seguridad puede que el ordenador se bloquee.

También nos puede interesar que nuestro programa termine y pase el control a otro programa de QBasic, por ejemplo en el caso de que sea un menú. Para hacerlo haríamos esto.

```
RUN "Programa.bas"
```

Al ejecutar esta orden nuestro programa termina inmediatamente y QBasic carga e inicia el otro programa basic. Esta orden equivale a que nosotros detengamos nuestro programa inmediatamente, en el editor abramos el otro programa y lo ejecutemos pulsando F5.
Si al usar la orden RUN todavía no habíamos guardado los cambios en el código fuente aparecerá un mensaje del editor pidiéndonos que lo hagamos, esto no es un error, cuando lo hagamos se continuará con el otro programa.
Si el programa no existe se producirá un error de tiempo de ejecución.
Si el programa no es un programa basic se producirá un error de compilación al intentar iniciarlo, nuestro programa anterior para entonces ya habrá terminado.

Al pasar de un programa a otro se cierran los ficheros que hubiera abiertos y desaparecen todas las variables. En algunos casos, todavía más rebuscados, nos puede interesar seguir trabajando con los ficheros abiertos y con las variables globales que fueron declaradas con la instrucción COMMON. Para conseguir esto en vez de usar la instrucción RUN usaremos la instrucción CHAIN, que trabaja de forma idéntica, excepto en que conserva los ficheros abiertos y reconoce las variables declaradas con COMMON en programas anteriores.

Normalmente una buena jerarquía de procedimientos SUB será más recomendable y más estructurada que usar la orden RUN o CHAIN. Estas ordenes solo las deberemos usar en casos muy sencillos como un pequeño menú que nos abra varios programas. La orden CHAIN no la deberemos usar casi nunca y sobre todo no encadenar varios programas ya que perderemos completamente la pista de dónde declaramos las variables y abrimos los archivos, cosa que puede dar lugar a errores muy difíciles de solucionar, como ocurre siempre que no estructuramos bien los programas.

TEMA 2.1
DETECCIÓN DE TECLAS PULSADAS CON LA FUNCIÓN INKEY$

- 2.1.1 - Uso de la función INKEY$
- 2.1.2 - Reconocimiento de teclas especiales
- 2.1.3 - Hacer pausas en los programas

2.1.1 - USO DE LA FUNCIÓN INKEY$

La función INKEY$ nos devuelve el contenido del buffer del teclado, es decir, cual es la última tecla o combinación de teclas que se ha pulsado. Esta función es más primitiva que INPUT, pero no la pudimos ver en el tema de entrada y salida básica porque todavía no sabíamos cómo usar los bucles, algo imprescindible para usar esta función ya que el programa no se detiene como ocurre con la instrucción INPUT. También hay que tener en cuenta que no es algo básico de la teoría de programación, ya que en otros lenguajes puede que funcione de formas muy distintas.

Para usar la función lo que haremos es almacenar su valor en una variable, ya que al leer el valor de INKEY$ el carácter que contenga se borra del buffer de teclado y ya no lo veremos más.
Vamos con el primer ejemplo:

```
CLS
tecla$ = INKEY$
PRINT tecla$
```

Este ejemplo funciona, aunque al ejecutarlo no lo parezca. Lo que hace es borrar la pantalla, almacenar la última tecla pulsada (durante el tiempo de vida del programa) en la variable tecla$ y a continuación escribirla en pantalla. Lo que pasa es que como el programa no se detiene no nos da tiempo a pulsar ninguna tecla y no aparece nada en pantalla, salvo el conocido rótulo "Presione cualquier tecla y continúe".

Para solucionar esto tenemos que recurrir a los bucles y programar lo que se conoce como "mecanismo de espera activa". Allá vamos, el siguiente fragmento de código será de los más usados en juegos y programas interactivos hechos con QBasic:

```
DO
        tecla$=INKEY$
LOOP WHILE tecla$=""
```

Al hacer esto se entra en un bucle que lee continuamente la entrada de teclado y la almacena en la variable tecla$. No salimos del bucle mientras no pulsemos nada y la variable tecla$ siga estando vacía. Una vez que pulsemos una tecla

esta quedará almacenada en la variable para usarla más adenante donde haga falta.

Esto ya funciona bien, pero todavía no es infalible. Puede ocurrir que en el buffer del teclado quede alguna pulsación de tecla residual de alguna instrucción de entrada anterior y que al llegar aquí se salte la espera. Para que esto no ocurra tendremos que "limpiar el buffer del teclado" antes de entrar al bucle. Para hacerlo usaremos otro bucle que lee las posibles teclas (y las va borrando) del buffer hasta que se quede vacío. Vamos a ver un ejemplo completo que espera a que pulsemos una tecla y la saca por pantalla. El primer bucle, de tipo WHILE, corresponde a la limpieza del buffer y el segundo, de tipo DO..LOOP, a la espera activa.

```
CLS
WHILE INKEY$<>""
WEND
DO
       tecla$=INKEY$
LOOP WHILE INKEY$=""
PRINT tecla$
```

Como se puede ver, este bucle de limpieza es un bucle WHILE, es decir, un bucle rechazable. Si el buffer ya está vacío ni siquiera entramos, en caso contrario entramos y leemos y borramos la tecla que pudiera tener antes de ir al otro bucle de reconocimiento de teclas.

2.1.2 - RECONOCIMIENTO DE TECLAS ESPECIALES

Como hemos dado por supuesto en los ejemplos anteriores las teclas se almacenan en el buffer del teclado usando el código ASCII o la representación del carácter que llevan. Por ejemplo la "a minúscula" se almacena como "a" o su código ASCII que es el 97, la "a mayúscula" lo hace como "A" o 65, el número cinco como "5" o 53, el punto como "." o 46 , el espacio en blanco como " " o 32 y así con todos los caracteres que tenemos en la sección alfanumérica del teclado.

Para trabajar con los códigos ASCII usaremos la conocida función CHR$(), de esta forma es lo mismo decir...

```
IF tecla$ = "A" THEN
```

...que...

```
IF tecla$ = CHR$(65) THEN
```

De esta forma también podemos identificar otras teclas solo con conocer su código ASCII. Estos son algunos de los más usados:

```
Escape.....: 27
Enter......: 13
Tabulador..: 9
```

```
Retroceso..: 8
```

También es posible identificar las teclas especiales de flechas (Imprescindible para los juegos), las teclas de función y las otras como Inicio, Fin, Insertar, etc. Al pulsar estas teclas se almacenan en el buffer del teclado dos caracteres, siendo el primero el CHR$(0). Para identificar estas teclas no hay más que usar en las comprobaciones CHR$(0)+El carácter que tengan asignado, aquí va una lista de los más usados:

```
Flecha arriba......: CHR$(0)+"H"
Flecha abajo.......: CHR$(0)+"P"
Flecha izquierda...: CHR$(0)+"K"
Flecha derecha.....: CHR$(0)+"M"

F1.................: CHR$(0)+";"
F2.................: CHR$(0)+"<"
F3.................: CHR$(0)+"="
F4.................: CHR$(0)+">"

F5.................: CHR$(0)+"?"
F6.................: CHR$(0)+"@"
F7.................: CHR$(0)+"A"
F8.................: CHR$(0)+"B"

F9.................: CHR$(0)+"C"
F10................: CHR$(0)+"D"
F11................: CHR$(0)+CHR$(133)
F12................: CHR$(0)+CHR$(134)

Insertar...........: CHR$(0)+"R"
suprimir...........: CHR$(0)+"S"
Inicio.............: CHR$(0)+"G"
Fin................: CHR$(0)+"O" (Letra o mayúscula)
Retroceder página..: CHR$(0)+"I" (Letra i mayúscula)
Avanzar página.....: CHR$(0)+"Q"
```

Podemos ver una lista completa en la ayuda en pantalla de QBasic en el apartado llamado "Códigos de exploración del teclado". Aquí también aparecen los códigos de teclas combinadas como por ejemplo CONTROL+Z.

Vamos a ver un ejemplo final de un programa que reconozca todas las teclas alfanuméricas normales además de Enter, Escape y las flechas que son especiales. En temas siguientes cuando hagamos programas más "bonitos" usaremos ampliamente el reconocimiento de teclas.

```
CLS
PRINT "Pulsa las teclas que quieras, escape para salir"
PRINT
DO
   WHILE INKEY$ <> "": WEND 'Limpia buffer de entrada
   DO
      tecla$ = INKEY$
   LOOP WHILE tecla$ = ""
   SELECT CASE tecla$
      CASE CHR$(0) + "H": PRINT "Has pulsado la flecha arriba"
      CASE CHR$(0) + "P": PRINT "Has pulsado la flecha abajo"
      CASE CHR$(0) + "K": PRINT "Has pulsado la flecha izquierda"
```

```
      CASE CHR$(0) + "M": PRINT "Has pulsado la flecha derecha"
      CASE CHR$(13): PRINT "Has pulsado Enter"
      CASE CHR$(27): PRINT "Has pulsado Escape, adios"
      CASE " ": PRINT "Has pulsado la barra de espacio"
      CASE ELSE: PRINT "Has pulsado la tecla " + tecla$
   END SELECT
LOOP UNTIL tecla$ = CHR$(27)
```

Observa que este programa no reconoce las teclas de función, entre otras, para hacerlo no hay más que añadir las instrucciones CASE con los códigos correspondientes.

2.1.3 - HACER PAUSAS EN LOS PROGRAMAS

Una forma muy sencilla de detener la ejecución de un programa, por ejemplo para que el usuario pueda leer algo es usar la instrucción SLEEP. Al usar esta instrucción el programa se detiene hasta que el usuario pulse una tecla. Si añadimos un número a continuación, por ejemplo:

```
SLEEP 10
```

Ocurrirá que el programa se detendrá por un máximo de 10 segundos, continuando pasado este tiempo automáticamente aunque el usuario no pulse ninguna tecla.

Esta instrucción funcionará en la mayoría de los casos, pero puede ocurrir que alguna vez no llegue a detener el programa porque hay datos residuales en el buffer de entrada, en estos casos podremos usar lo que hemos visto en el apartado anterior: Limpieza del buffer+Espera activa, es decir esto:

```
WHILE INKEY$<>"":WEND
DO:LOOP WHILE INKEY$=""
```

Esta estructura la podemos ampliar para hacer una pregunta al usuario y dejarle responder Sí o No pulsando S o N, por ejemplo.

```
WHILE INKEY$<>"":WEND
DO
       tecla$=INKEY$
LOOP UNTIL tecla$="S" OR tecla$="N"
```

Aquí entramos en el bucle de espera activa y no salimos hasta que el usuario pulse la S o la N, y después más adelante en el programa viendo el valor de tecla$ ya decidiríamos entre hacer algo o no. Pero todavía hay un problema, este ejemplo tal como está sólo reconoce la S y la N mayúsculas, por lo que si el usuario tiene el "Bloqueo de mayúsculas" desactivado puede que no se le ocurra como seguir. Para arreglar el problema usaremos la función UCASE$ que convierte todas las letras (Menos la Ñ y las vocales acentuadas) a mayúsculas. Vamos con el ejemplo completo:

```
CLS
```

```
PRINT "¿Quieres hacer no se qué? S/N: "
WHILE INKEY$<>"":WEND
DO
       tecla$=UCASE$(INKEY$)
LOOP UNTIL tecla$="S" OR tecla$="N"

IF tecla$="S" THEN
       PRINT "Has dicho que sí"
ELSE
       PRINT "Has dicho que no"
END IF
```

Ampliando más este ejemplo podríamos construir un menú "cutre". Si dejamos al usuario la posibilidad de pulsar más teclas, normalmente números, y después los comprobamos con un SELECT CASE podríamos decidir entre hacer varias cosas. Es una estructura similar a la usada en el ejemplo de reconocer teclas del apartado anterior.

TEMA 2.2
NÚMEROS ALEATORIOS Y CONTROL DE TIEMPO

- **2.2.1 - Números aleatorios. Función RND**
- **2.2.2 - Control de tiempo**

2.2.1 - MÚMEROS ALEATORIOS. FUNCIÓN RND

Cuando diseñamos un programa informático lo que intentamos es resolver un problema obteniendo un resultado lo más exacto y fiable posible, usando la gran precisión de los cálculos que hacen los ordenadores. De hecho, una de las normas básicas para asegurar que un algoritmo funciona correctamente es poder asegurar y demostrar que para unos mismos datos de entrada los resultados o datos de salida van a ser siempre los mismos.

En algunos casos como son los videojuegos nos puede interesar que suceda justamente lo contrario de lo que se ha dicho en el párrafo anterior, es decir, que cada vez que ejecutemos el programa sucedan cosas distintas y aparentemente impredecibles. Otras aplicaciones donde tendría que suceder esto serían programas simuladores de cálculos y fenómenos científicos y algunas aplicaciones matemáticas o estadísticas especiales.

Para poder llegar a estas situaciones nos encontramos con el terrible problema de conseguir que el ordenador (tan perfecto, exacto, preciso, etc...) sea capaz de generar números de cualquier manera sin responder a ningún orden aparente. Esto técnicamente es imposible y por lo tanto en vez de hablar de números aleatorios sería más correcto llamarlos números pseudo-aleatorios porque no son completamente impredecibles, pero lo son casi, lo suficiente como para resolver nuestras necesidades.

Para generar estos números (pseudo-) aleatorios el ordenador puede usar mecanismos tales como medir el tiempo entre dos pulsaciones de teclas, o el necesario para enviar o recibir un dato a través de una red. Como esto puede ser difícil de conseguir en algunos casos, es mucho más cómodo usar lo que se conoce como "generador de números aleatorios" que es una formula matemática que al aplicarla repetidas veces pasándole los valores anteriores nos va devolviendo una serie de números sin orden aparente. Una de estas fórmulas se llama "Generador de Marsaglia" y viene a ser algo así como:

$$z_N = (2111111111 * z_{N-4} + 1492 * z_{n-3} + 1776 * z_{n-2} + 5115 * z_{n-1} + C) \text{ MOD } 232$$

$$C = \text{FLOOR}((2111111111 * z_{N-4} + 1492 * z_{n-3} + 1776 * z_{n-2} + 5115 * z_{n-1} + C) / 232)$$

Esta fórmula nos devolvería una serie de números que no empezaría a repetirse hasta que lo hayamos calculado 3 * 10 elevado a 47 veces, un 3 con 47 ceros detrás, más que suficiente. Pero que nadie se asuste, de hecho no se ni si están bien copiados los números. En QBasic (y en otros lenguajes de programación) no nos hace falta programar tantos relíos, basta con usar el valor que nos

devuelva la función RND cada vez que queramos obtener un número aleatorio. Vamos con un ejemplo de los más sencillo del curso de programación:

```
CLS
PRINT RND
```

Al ejecutarlo se escribiría en pantalla el número aleatorio que devuelva la función RND, un valor decimal (casi) impredecible, por ejemplo:

```
.7055475
```

Ya lo tenemos ahí, un número generado por el ordenador sin responder a ningún criterio aparente (Lleva un cero delante, pero QBasic se lo quita al escribirlo en la pantalla y solo se ve el punto decimal), pero todavía hay un problema, ejecuta el programa varias veces y mira el resultado.
Puedes comprobar que siempre sale el mismo número, ya no es tan impredecible. Esto se debe a que no hemos inicializado el generador de números aleatorios de QBasic.

Si tratas de descifrar algo de la fórmula de Marsaglia que hay más arriba observarás que para calcular un número pseudo-aleatorio se necesitan algunos de los calculados anteriormente. Esto va muy bien si ya los hemos calculado, pero si es la primera vez no los tenemos. En este caso tendremos que usar lo que se conoce como "Semilla", que es un número arbitrario usado para que empiece a funcionar el generador. Si este número es el mismo siempre los números aleatorios devueltos serán siempre una misma serie, por lo tanto hay que usar como semilla un número lo más "aleatorio" posible como por ejemplo el número de segundos que han pasado a partir de las doce de la noche, cosa que nos devuelve la función TIMER. Para inicializar el generador de números aleatorios usaremos la instrucción RANDOMIZE seguida de la semilla, normalmente TIMER. Vamos con el ejemplo completo:

```
RANDOMIZE TIMER
CLS
PRINT RND
```

Ahora sí que cada vez que ejecutemos el programa tendremos un número distinto. El valor solo se repetiría si ejecutamos el programa otro día a la misma hora, mismo minuto, mismo segundo. Algo difícil de conseguir y un riesgo aceptable en programas que no necesiten elevadas medidas de seguridad como los nuestros, pero no en otras aplicaciones como pudieran ser máquinas recreativas. De hecho, en sistemas más serios como Lunux se puede observar como al apagar el ordenador se guarda en algún sitio la semilla aleatoria para poderla seguir usando la próxima vez.

La instrucción RANDOMIZE TIMER es algo así como remover las bolas en un sorteo de lotería, debemos usarla una vez al principio de los programas que usen la función RND, pero no es necesario usarla más, de hecho repetirla en sitios como dentro de un bucle podría resultar contraproducente.

El valor devuelto por RND va a ser un número decimal de precisión sencilla mayor o igual que cero y menor que uno, es decir, alguna vez podría salir el cero, pero nunca saldrá el uno. Este valor lo podemos multiplicar o redondear

para adaptarlo a nuestras necesidades.
Vamos con un programa que contiene dos ejemplos:

```
CLS
RANDOMIZE TIMER
valorDado = INT(RND*6)+1
PRINT "Hemos lanzado un dado y ha salido"; valorDado
IF RND > 0.5 THEN
       PRINT "Hemos lanzado una moneda y ha salido 'cara'"
ELSE
       PRINT "Hemos lanzado una moneda y ha salido 'cruz'"
END IF
```

Que daría este resultado (Con estos u otros valores):

```
Hemos lanzado un dado y ha salido 4
Hemos lanzado una moneda y ha salido 'cruz'
```

En el primer caso multiplicamos el valor de RND por seis, ya tenemos un número decimal que puede ir desde 0 a 5.999 (Nunca 6). A continuación lo redondeamos hacia abajo usando la función INT, ya tenemos un entero entre 0 y 5. Como queremos un valor posible entre 1 y 6, que son las caras que tienen los dados, no hay más que sumarle 1.
En el segundo caso lo que hacemos es comprobar el valor de RND para hacer una cosa u otra. Como en este caso queremos que las dos partes del IF tengan las mismas posibilidades ponemos el punto de comprobación en la mitad, en 0.5. Si quisiéramos variar las posibilidades no tendríamos más que cambiar ese número. Si queremos más de dos posibilidades podemos usar un SELECT CASE con varios intervalos.

Por último vamos a ver como conseguir que RND nos repita el último número que generó sin tener que grabarlo en ninguna variable. Para conseguirlo bastaría con escribir RND(0) en vez de RND. Vamos con un ejemplo:

```
CLS
RANDOMIZE TIMER
PRINT "Un número aleatorio.............:"; RND
PRINT "Repitamos el mismo número.......:"; RND(0)
PRINT "Otra vez más....................:"; RND(0)
PRINT "Ahora un número aleatorio nuevo.:"; RND
PRINT "Vamos a sacarlo otra vez........:"; RND(0)
```

El resultado podría ser (con otros números):

```
Un número aleatorio.............: .2051972
Repitamos el mismo número.......: .2051972
Otra vez más....................: .2051972
Ahora un número aleatorio nuevo.: .1969156
Vamos a sacarlo otra vez........: .1969156
```

Como se puede ver, si llamamos a RND sin argumentos, como hemos hecho hasta ahora, nos da un número aleatorio, pero si lo hacemos pasándole como argumento el cero entre paréntesis (Tiene que ser obligatoriamente el cero) lo que haces es repetir el mismo número en vez de calcular uno nuevo, que sería el siguiente de la serie.

Si usamos RND(0) por primera vez en el programa, sin haber usado antes RND, no pasa nada, tendremos un número aleatorio.
Esto no se utiliza demasiado, pero en algún caso podemos ahorrarnos una variable y una instrucción de asignación si queremos usar el mismo número en varios sitios.

2.2.2 - CONTROL DE TIEMPO

El ordenador lleva instalado un reloj digital que usa para muchas cosas, como por ejemplo para almacenar en los directorios la fecha y hora en que se guardó cada archivo. Por supuesto nosotros también podemos utilizar este valor en nuestros programas.

La forma más sencilla de obtener la hora ya la hemos visto en el apartado anterior. Es usar el valor devuelto por la función TIMER, un entero largo con los segundos que han pasado desde las doce de la noche. Con esta función ya podemos hacer un programa que nos salude de distinta forma dependiendo del momento del día que sea. Allá vamos...

```
CLS
SELECT CASE TIMER
       CASE IS < 28800: PRINT "Buenas madrugadas"
       CASE 28801 TO 43200: PRINT "Buenos días"
       CASE 43201 TO 72000: PRINT "Buenas Tardes"
       CASE ELSE: PRINT "Buenas noches"
END SELECT
```

El resultado, si ejecutamos el programa a las diez de la mañana sería:

```
Buenos días
```

Es una forma de que nuestro programa sea algo "inteligente", aunque todavía se podría mejorar (mucho). Lo que hemos hecho es ver cuantos segundos han pasado a las ocho de la mañana, a mediodía y a las ocho de la tarde para definir los intervalos, nada más.

También podemos usar la función TIMER para cronometrar el tiempo que tarda en pasar algo. Bastaría con almacenar la hora en una variable antes de empezar, hacer lo que sea, y al terminar almacenar la hora en otra variable y restarlas. Obtendríamos el número de segundos. Vamos a cronometrar lo que tarda el ordenador en escribir en pantalla los números del 1 al 10.000. Allá vamos...

```
CLS
inicio=TIMER
FOR n = 1 TO 10000
       PRINT n
NEXT
final = TIMER
tiempo =  final - inicio
PRINT "Ha tardado"; tiempo; "segundos"
```

Este programa, al final de la tira de 10.000 números nos daría un mensaje con el número de segundos que ha tardado.

Todo funcionaría bien excepto si durante la ejecución del programa han dado las doce de la noche, obteniéndose en este caso un número negativo. Para solucionar el problema habría que colocar la siguiente línea justo antes de sacar el mensaje. Lo que hacemos es sumarle todos los segundos de un día si ha dado negativo.

```
IF tiempo < 0 THEN tiempo = tiempo + 86400
```

Ahora vamos a recordar dos funciones que nos devuelven la fecha y la hora en modo texto, son DATE$ y TIME$.

```
CLS
PRINT "Hoy es "; DATE$
PRINT "Son las "; TIME$
```

Esta daría como resultado algo como:

```
Hoy es 01-20-2006
Son las 10:42:35
```

Con la hora no hay nada que aclarar, pero la fecha está en formato americano, primero el mes, después el día y al final el año. El mes y el día siempre con dos cifras y el año con cuatro. Estas funciones pueden ser más intuitivas pero para manejar las cifras las tenemos que separar usando la función MID$. Vamos con el ejemplo:

```
CLS
PRINT "El día es "; MID$(DATE$, 4, 2)
PRINT "El mes es "; MID$(DATE$, 1, 2)
PRINT "El año es "; MID$(DATE$, 7, 4)
PRINT
PRINT "La hora es "; MID$(TIME$, 1, 2)
PRINT "Los minutos son "; MID$(TIME$, 4, 2)
PRINT "Los minutos son "; MID$(TIME$, 7, 2)
```

Y el resultado podría ser:

```
El día es 20
El mes es 01
El año es 2006

La hora es 10
Los minutos son 42
Los minutos son 35
```

Muy sencillo. Lo único que hemos hecho es "recortar" con la función MID$ el trozo de cadena donde está cada cosa, que siempre va a estar en el mismo sitio, y separarlo para hacer con él lo que queramos. Y si nos hiciera falta sumarlos o compararlos podríamos convertirlos a enteros usando la función VAL.

Aplicando esto mismo, vamos con un ejemplo de una función a la que vamos a llamar FECHA$ y nos va a devolver la fecha actual como la usamos normalmente en castellano: Día, mes y año separados por barras.

```
FUNCTION FECHA$
        aux$ = MID$(DATE$, 4, 2)
        aux$ = aux$ + "/" + MID$(DATE$, 1, 2)
        aux$ = aux$ + "/" +MID$(DATE$, 7, 4)
        FECHA$ = aux$
END FUNCTION
```

Al llamarla desde cualquier parte de nuestro programa nos devolvería, en formato cadena algo como 20/01/ 2006, que queda más bonito que 01-20-2006.

Para terminar vamos a ver como cambiar la fecha y hora del sistema desde nuestro programa. Algo que no es habitual ni recomendable, pero que en algún caso nos puede hacer falta.
Sería usando las instrucciones, no funciones sino instrucciones, DATE$ y TIME$ y asignándoles cadenas que contengan la fecha o la hora en el formato adecuado. Vamos con unos ejemplos:

```
DATE$ = "04-20-2006"
DATE$ = "04-20-92"
DATE$ = "04-06-1994"

TIME$ = "10:24:35"
TIME$ = "10:24"
TIME$ = "10"
```

La fecha podemos establecerla poniendo siempre el mes primero y después el día y el año, este último dato puede tener dos o cuatro cifras. En el último ejemplo podríamos dudar entre si la fecha es el 4 de junio o el 6 de abril, se almacenaría este último, 6 de abril. Cuidado con esto, siempre mes, día, año.

En el caso de la hora si no ponemos los minutos o los segundos se entenderán que son 00.

Si intentamos asignar una fecha o una hora no válida se producirá un error de tiempo de ejecución. Normalmente no es recomendable modificar estas cosas desde los programas, si lo hacemos mal y no nos damos cuenta estaremos un tiempo usando el ordenador con el reloj mal, provocando problemas en los directorios, los antivirus, el planificador de tareas programadas, etc.

TEMA 2.3
MÁS SOBRE LAS CADENAS DE CARACTERES

- **2.3.1 - Acerca de las cadenas de caracteres**
- **2.3.2 - La función e instrucción MID$**
- **2.3.3 - Otras funciones de manejo de cadenas**

2.3.1 - ACERCA DE LAS CADENAS DE CARACTERES

En QBasic el manejo de cadenas de caracteres es extremadamente sencillo. En otros lenguajes de programación como es el caso de C las cadenas se tratan como un array de datos de tipo carácter (un tipo que no tenemos en QBasic) de longitud aproximadamente igual al número de caracteres de la cadena, y por lo tanto su longitud máxima está siempre limitada. También esta forma de trabajar implica que para hacer algo tan simple como una asignación de cadenas en esos lenguajes haya que recurrir a funciones especiales del lenguaje y no podamos usar directamente el operador de asignación, el signo igual, como si se tratara de números.

En este tema vamos a ver con detalle algunas funciones de QBasic relacionadas con cadenas que nos van a servir para manejar los datos de este tipo que nos hagan falta en nuestro programa.

Vamos a recordar un poco lo que ya sabemos de las cadenas:

Al igual que ocurre con las otras variables, no es necesario declararlas. Basta con usar un nombre de variable terminado por el carácter dólar ($). En este caso tendremos una cadena de texto de longitud variable.

En algunos casos nos puede interesar que las cadenas sean de longitud fija para ahorrar memoria (Unos 10 bytes por cadena). En este caso bastaría con declararlas usando:

```
DIM nombreVariable AS STRING * 25
```

Dónde el número que va a continuación del asterisco es la longitud máxima de la cadena. Es especialmente recomendable declarar las cadenas como de longitud fija en el caso de los arrays. Por ejemplo si hacemos:

```
DIM tabla (1 TO 40, 1 TO 3) AS STRING * 5
```

Nos estamos ahorrando unos 120 bytes en todo el array, además de que el manejo de cadenas de tamaño fijo por parte del ordenador es algo más rápido por tratarse de estructuras estáticas.

En el caso de las estructuras de datos definidas por el usuario (tipos registro) que incluyan cadenas, estas siempre tendrán que declararse con una longitud determinada.

2.3.2 - LA FUNCIÓN E INSTRUCCIÓN MID$

Una de las funciones más útiles que incluye QBasic para el manejo de cadenas de caracteres es MID$. Ya la hemos visto anteriormente en varios ejemplos, pero vamos a hacerlo ahora con más detalle. Esta es su sintaxis.

```
MID$("cadena", inicio, longitud)
```

Esta función lo que hace es extraer una porción de una cadena. Le debemos pasar tres argumentos:

- "cadena" es una cadena de caracteres entre comillas o bien el nombre de una variable de cadena cuyo valor va a ser utilizado por la función, o una expresión que devuelva como resultado una cadena.
- inicio es la posición del primer carácter que queremos sacar de la cadena, el carácter de más a la izquierda es el 1. Si este valor es superior al tamaño de la cadena, la función MID$ devolverá como resultado una cadena vacía.
- longitud es el tamaño del trozo de cadena que queremos extraer. Si se sobrepasa el final de la cadena no ocurre nada, sólo se devolverá lo que se pueda de antes del final.

Vamos con unos ejemplos:

```
CLS
miCadena = "Hecho en Ronda, Ciudad Soñada"
PRINT MID$(miCadena, 1, 5)
PRINT MID$(miCadena, 5, 3)
PRINT MID$(miCadena, 15, 1)
PRINT MID$(miCadena, 26, 2)
PRINT MID$(miCadena, 300, 1)
PRINT MID$(miCadena, 10, 3000)
```

El resultado sería:

```
Hecho
o e
,
ña

Ronda, Ciudad Soñada
```

Esta función nos puede servir para extraer un determinado carácter de una cadena. Algo que puede parecer trivial, pero que nos va a simplificar mucho determinados problemas.

Vamos con un ejemplo tonto:

```
CLS
miCadena = "LMXJVSD"
```

```
INPUT "Escribe un número del 1 al 7: ", num
PRINT "El"; num; "º carácter de "; miCadena; " es "; MID$(miCadena,
num, 1)
```

Un resultado posible sería:

```
Escribe un número del 1 al 7: 4
El 4 º carácter de LMXJVSD es J
```

Muy sencillo. Sólo decir que no hemos depurado el dato de entrada y que si escribimos un cero o negativo habrá un error.

Vamos con otro ejemplo más útil que utiliza esta misma técnica para determinar la letra del DNI. Para hacer esto lo que hay que hacer es comparar el resto del número dividido entre 23 con una serie de letras ordenadas de una forma característica y dar la que corresponda.

```
CLS
INPUT "Escribe el número del DNI: ", dni&
PRINT "La letra es: ": MID$("TRWAGMYFPDXBNJZSQVHLCKE", (dni& MOD
23) + 1, 1)
```

El resultado podría ser:

```
Escribe el número del DNI: 74926208
La letra es: M
```

Podemos ver que el cálculo se hace en una sola linea de código usando la instrucción MID$.

- El primer argumento que le pasamos a MID$, la cadena de caracteres, es la tira de letras ordenadas de forma característica.
- La posición de inicio es el cálculo propiamente dicho. Le sumamos uno porque la primera letra es la 1 y no la 0.
- El tamaño es 1 porque siempre queremos una y sólo una letra.

Es importante ver que la variable dónde guardaremos el número debe ser de tipo entero largo. Si la usamos como de tipo real (sin poner el &) se redondeará y no saldrá bien el cálculo.

Con esta técnica el algoritmo ha resultado sumamente corto. A lo mejor sería más sencillo o intuitivo haberlo resuelto usando un vector de caracteres para poder acceder a sus posiciones individuales, o bien un SELECT CASE. Con cualquiera de estas soluciones el listado del programa hubiera sido mucho más largo. Tendríamos 23 posibilidades en el SELECT CASE o bien 23 asignaciones al vector.

Este problema del DNI es muy frecuente. Le falta depurar los datos de entrada y convertirlo en forma de función, para que sea mucho más portable.

Para terminar con MID$ hay que decir que además de ser una función, se puede usar también como instrucción para modificar la propia cadena.

La sintaxis sería

```
MID$( VariableCadena$, inicio, longitud) = Cadena$
```

El funcionamiento es parecido, salvo unas cuantas diferencias:

- El primer argumento tiene que ser una variable de cadena y no un literal entre comillas ni una expresión.
- El valor de inicio no puede ser mayor que la longitud de la cadena, en este caso se produciría un error de tiempo de ejecución.

Vamos con un ejemplo:

```
miCadena$ = "Hecho en Ronda"
PRINT miCadena$
MID$(miCadena$, 10, 5) = "Soria"
PRINT miCadena$
MID$(miCadena$, 10, 5) = "Sevilla"
PRINT miCadena$
MID$(miCadena$, 7, 8) = "aquí"
PRINT miCadena$
```

El resultado sería...

```
Hecho en Ronda
Hecho en Soria
Hecho en Sevil
Hecho aquíevil
```

Hay que tener en cuenta que la longitud del tramo de cadena a reemplazar queda definido por el valor del tercer parámetro, y no por el tamaño de la expresión de cadena que asignamos. Si este es mayor se cortará, y si no llega sólo se usará lo que haya, quedando lo demás como estaba. De esta misma forma la cadena nunca aumentará o disminuirá su longitud total usando esta función. Como mucho podremos disminuir su longitud aparente usando espacios en blanco, pero realmente seguirán estando ahí formando parte de la cadena.

MID$ se usa mucho más como función que como instrucción. Hay que tener claro para que sirve cada cosa y cuando se está usando una u otra. Como función va siempre a la derecha del operador de asignación o dentro de una expresión, y como instrucción va siempre al principio de la línea de código.

2.3.3 - OTRAS FUNCIONES DE MANEJO DE CADENAS

QBasic nos ofrece varias funciones útiles para hacer operaciones con cadenas de caracteres. Una de las más útiles es MID$ a la que hemos dedicado el apartado anterior completo. Aquí van otras:

Empecemos con dos funciones algo parecidas a MID$, pero menos avanzadas.

```
LEFT$("cadena", numCaracteres)
RIGTH$("cadena", numCaracteres)
```

La primera de ellas devuelve un determinado número de caracteres del principio de la cadena, de la izquierda, y la otra los devuelve del final, de la derecha. Siempre en el mismo orden en que están en la cadena. Se puede hacer referencia tanto a una cadena literal entre comillas o a una variable o expresión de tipo cadena. Si el número de caracteres especificado es mayor que la longitud total de la cadena se devolverá la cadena entera. Si es cero se devolverá una cadena vacía, y si es negativo habrá un error.

```
cadena$ = "Hecho en Ronda"
PRINT LEFT$(cadena$, 5)
PRINT RIGHT$(cadena$, 5)
```

Daría como resultado:

```
Hecho
Ronda
```

Ahora vamos con otras que convierten a mayúsculas y minúsculas...

```
UCASE$("Cadena")
LCASE$("Cadena")
```

UCASE$ convierte todas las letras que haya en la cadena, variable de cadena o expresión, a mayúsculas (Upper Case), y LCASE$ a minúsculas (Lower Case). Es importante tener en cuenta que no se reconocen como letras ni los acentos ni la eñe ni la u con diéresis (ü) y por lo tanto no se convierten correctamente. Vamos con un ejemplo.

```
cadena$ = "Una cigüeña en un balcón de África"
PRINT UCASE$(cadena$)
PRINT LCASE$(cadena$)
```

Daría:

```
UNA CIGüEñA EN UN BALCóN DE ÁFRICA
una cigüeña en un balcón de África
```

Ahora vamos con otras dos funciones que eliminarán los espacios en blanco que pueda haber en los extremos de una cadena.

```
LTRIM$("Cadena")
RTRIM$("Cadena")
```

LTRIM$ elimina los espacios que puede haber delante (A la izquierda, left) y RTRIM$ los que pueda haber por el final (A la derecha, right).

Esta última función es muy útil para trabajar con datos almacenados en cadenas de longitud fija (Como las que se usan en los tipos de datos definidos por el usuario para los registros). Estas cadenas siempre van rellenas con espacios hasta ocupar su longitud total, y si las manejamos con todos estos

espacios pueden pasar cosas como que fallen las comparaciones o que pasen cosas imprevistas en los diseños de pantallas.

Ambas funciones pueden ser útiles para depurar datos introducidos por teclado en los que el usuario haya podido escribir espacios inútiles antes o después. Vamos con un ejemplo:

```
cadena$ = "          Hola!          "
PRINT "*"; LTRIM$(cadena$); "*"
PRINT "*"; RTRIM$(cadena$); "*"
```

Y el resultado:

```
*Hola!          *
*          Hola!*
```

En Visual Basic se dispone de una función TRIM$ que elimina los espacios tanto delante como detrás de la cadena. Aquí no la tenemos, pero su programación sería muy sencilla construyendo una nueva función a partir de estas dos.

Ahora vamos con una función que nos devuelve una cadena llena con el número de espacios que le digamos

```
SPACE$(num)
```

Puede parecer inútil, pero nos ayudará bastante en el diseño de pantallas. Por ejemplo:

```
PRINT "Hola"; SPACE$(40); "Que hay"
```

Daría:

```
Hola                                        Que hay
```

O el siguiente ejemplo más elaborado:

```
CLS
FOR n = 0 TO 5
       PRINT SPACE$(5 - n); "/"; SPACE$(n + n); "\"
NEXT
```

Dibujaría esta figura:

Vamos con otra función parecida, pero que en vez de devolver una cadena de espacios la devuelve del carácter que nosotros le digamos:

```
STRING(Longitud, "carácter")
```

o bien

```
STRING(Longitud, Código-ASCII)
```

Como se puede ver podemos especificar el carácter que queremos usando una cadena o bien usando el número de su código ASCII. Esto es especialmente útil cuando queremos dibujar un carácter que no aparece en el teclado.

Vamos con unos ejemplos:

```
CLS
PRINT STRING$(10, "*")
PRINT STRING$(5, 60)
PRINT STRING$(15, "RONDA")
PRINT STRING$(8, 126)
```

Que daría

```
**********
<<<<<
RRRRRRRRRRRRRR
~~~~~~~~
```

Al igual que la anterior, esta función nos será muy útil para construir diseños de pantallas, pero hay que tener cuidado de no confundir su nombre STRING$ con el de tipo de datos cadena que es STRING sin el dólar detrás.

Vamos ahora con otra función que en vez de devolvernos la cadena con cierta modificación nos va a devolver un número que corresponde a la longitud (Número de caracteres) de la cadena.

```
LEN("cadena")
```

Vamos con un ejemplo que en combinación con la función STRING$ que acabamos de ver nos subraye una palabra usando guiones.

```
CLS
INPUT "Escribe algo: "; cad$
PRINT "              "; STRING$(LEN(cad$), "-")
```

Daría algo como:

```
Escribe algo: YA ESCRIBO ALGO
              ---------------
```

Esta función LEN también nos devuelve el espacio en bytes que ocupa en memoria cualquier variable. Basta con pasarle como parámetro el nombre de una variable que no sea de cadenas.

A esta categoría de funciones de manejo de cadenas habría que añadir otras que ya hemos visto como son DATE$ y TIME$, así como algunas menos utilizadas como HEX$ y OCT$ que convierten un número a sistemas de numeración en base 16 y en base 8 respectivamente, pero ya que el resultado es en forma de cadena no nos servirá para hacer ningún cálculo.

Combinando estas funciones podemos construir otras más potentes que nos hagan cosas como por ejemplo dibujar recuadros o centrar textos. Esto es lo que se verá en el tema siguiente de manejo de pantalla del texto.

TEMA 2.4
MANEJO DE LA PANTALLA EN MODO TEXTO

- **2.4.1 - Introducción a los interfaces de texto**
- **2.4.2 - Posicionar el cursor en la pantalla**
- **2.4.3 - Texto en colores**
- **2.4.4 - Redefinir colores**
- **2.4.5 - Caracteres semigráficos**
- **2.4.6 - Caracteres especiales de relleno**
- **2.4.7 - Caracteres no imprimibles**
- **2.4.8 - Esmáilis y ASCII-ART**

2.4.1 - INTRODUCCIÓN A LOS INTERFACES DE TEXTO

A estas alturas del siglo XXI casi todos los ordenadores personales utilizan modernos entornos gráficos como son Windows o los también conocidos KDE y GNOME en Linux. Estos sistemas pueden representar cualquier imagen a partir de pequeños puntos que pueden ser de uno de los millones de colores soportados. Para aplicaciones más sencillas, como es el caso del entorno de QBasic, se utiliza lo que se denomina Interfaz de texto, que es mucho más fácil de controlar por el ordenador y más que suficiente para determinadas aplicaciones.

En todos los ejemplos que hemos hecho hasta ahora hemos podido comprobar como los resultados se iban presentando en la pantalla de forma secuencial, es decir, unos debajo de otro conforme se iban generando. Cuando se alcanzaba la parte baja de la pantalla todo el contenido subía automáticamente para dejar una nueva línea vacía abajo. Algunas operaciones más "sofisticadas" que hemos llegado a hacer han sido borrar todo usando la instrucción CLS o encolumnar la presentación de algunos listados ajustando los argumentos que le pasamos a la instrucción PRINT. De todas formas los resultados siempre se han visto de color gris sobre fondo negro. Muy triste.

En este tema vamos a ver cómo se puede conseguir escribir exactamente en la posición de la pantalla que queramos, así como en distintos colores de letra y de fondo. También usaremos unos caracteres especiales conocidos como "Semigráficos" para dibujar líneas y recuadros que den más consistencia a nuestras presentaciones.

2.4.2 - POSICIONAR EL CURSOR EN LA PANTALLA

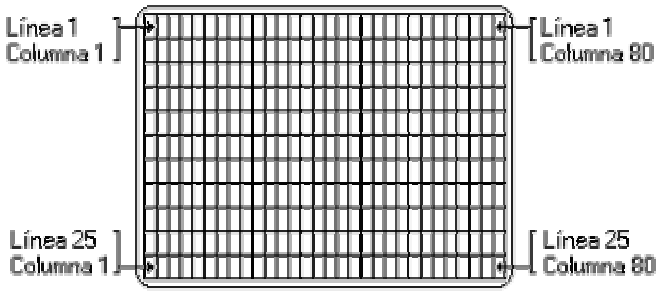
El cursor es un objeto que se puede mover por la pantalla y su posición indica el sitio exacto dónde se escribirá lo siguiente. En los procesadores de textos o en el propio editor de QBasic, es la "barrita" intermitente por donde van apareciendo las letras que vamos escribiendo en el teclado. En cualquier

programa en modo texto el cursor siempre va a existir y va a indicar el lugar dónde se va a escribir lo siguiente que haya que escribir, ya sea porque lo haga el usuario con el teclado o bien el propio programa atendiendo a sus instrucciones de salida.

En los programas de QBasic sólo es visible cuando se pide al usuario que escriba algo usando la instrucción INPUT. El resto del tiempo está ahí en su sitio, pero no lo vemos.

Cuando usamos cualquier instrucción PRINT sin punto y coma al final se escribe lo que sea y el cursor invisible pasa al principio de la siguiente línea a la espera de la siguiente instrucción de escritura. De esta forma todos los resultados se van mostrando uno debajo de otro línea por línea.

La pantalla normal en modo texto que usamos aquí está compuesta por 25 líneas de 80 caracteres de ancho.



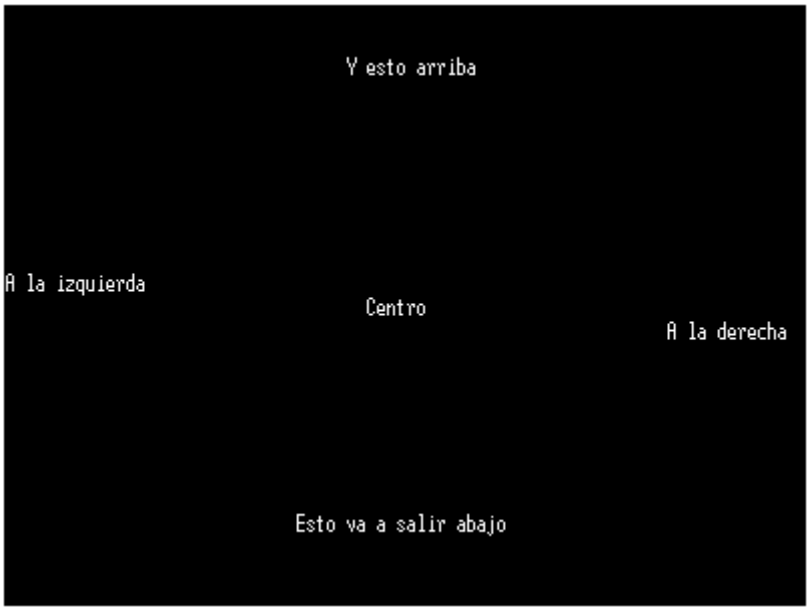
La primera línea, la de más arriba es la número 1 y la de abajo es la 25. La columna de más a la izquierda es la 1 y la de más a la derecha, la última, es la 80. Para llevar el cursor a cualquiera de estas posiciones y escribir en la parte de la pantalla que queramos no hay más que usar la siguiente instrucción:

```
LOCATE linea, columna
```

De esta forma tan sencilla, usando una instrucción LOCATE delante de cada PRINT o INPUT ya podemos controlar exactamente donde va a salir cada cosa. Vamos con un ejemplo:

```
CLS
LOCATE 22, 30
PRINT "Esto va a salir abajo"
LOCATE 3, 35
PRINT "Y esto arriba"
LOCATE 12, 1
PRINT "A la izquierda"
LOCATE 14, 67
PRINT "A la derecha"
LOCATE 13, 37
PRINT "Centro"
SLEEP
```

Y este sería el resultado. El recuadro representa al borde de la pantalla.



Cómo se puede ver, las frases ya no aparecen escritas de arriba a abajo en el mismo orden en que están escritas en el listado del programa. Cada una aparece en el sitio donde la anterior instrucción LOCATE ha llevado el cursor de texto. Para ajustar estas posiciones no hay más que variar los números de fila y columna de las instrucciones LOCATE. La instrucción SLEEP que aparece al final es para que el rótulo "Presione cualquier tecla y continúe." no aparezca hasta que no pulsemos una tecla. Esto será muy común para que no se nos estropeen los diseños de pantalla.

Otra cosa que nos va a estropear los diseños de pantalla va a ser el desplazamiento vertical que hace automáticamente el interfaz en modo texto para que conforme vamos escribiendo lo anterior se desplaza hacia arriba, como ha venido pasando en los programas que hemos hecho hasta ahora. Esto hace que si escribimos algo en la línea 24, las líneas 1 a 24 suban una posición, desapareciendo lo que haya en la primera línea. Y si escribimos algo en la línea 25, las líneas 1 a 24 subirán 2 posiciones, desapareciendo lo que hubiera escrito en las dos primeras líneas.

La solución más cutre para salvar este problema sería no escribir nunca en las líneas 24 y 25, pero hay otra mejor. Poner un punto y coma al final de todas las instrucciones PRINT o a continuación de la palabra INPUT en las instrucciones de entrada, siempre que estén situadas en una de estas dos últimas líneas. De esta forma evitaremos que al terminar la instrucción el cursor baje y se produzca el desplazamiento vertical automático.

Ahora ya podemos realmente escribir en cualquier parte de la pantalla usando la instrucción LOCATE.
Vamos con un ejemplo que aclare esto del desplazamiento automático.

```
CLS
LOCATE 1, 20: PRINT "Principio"
LOCATE 25, 20: PRINT "Final"
SLEEP
```

Si ejecutas este fragmento de código observarás que la palabra PRINCIPIO no aparece en la pantalla, ya que tras escribir en la línea 25 se produce este desplazamiento automático. Para solucionar el problema añadimos un punto y coma al final de la instrucción PRINT conflictiva:

```
CLS
LOCATE 1, 20: PRINT "Principio"
LOCATE 25, 20: PRINT "Final";
SLEEP
```

Y podrás observar como aparecen ambas palabras escritas en la pantalla, una arriba y otra abajo, justamente en las posiciones que hemos especificado. Problema solucionado.

Un último comentario. En los dos fragmentos anteriores has podido observar que las instrucciones LOCATE y PRINT van en la misma línea separadas por el carácter "Dos puntos" (:). Esto es común hacerlo para acortar el listado, ya que estas dos instrucciones casi siempre van a ir por parejas.

2.4.3 - TEXTO EN COLORES

Hasta ahora todos nuestros programas han dado los resultados en la pantalla usando letras de color gris claro sobre fondo negro. Para llorar, pero esto va a cambiar ahora mismo. El interfaz en modo texto nos ofrece la posibilidad de usar dieciséis colores de letra y ocho de fondo. No es que sea una maravilla tecnológica comparada con los millones de colores que usan los entornos gráficos actuales, pero para cumplir nuestros objetivos son más que suficientes.

Vamos a ver la paleta de colores predeterminada del modo de pantalla VGA:

Como se puede ver, cada color lleva un asociado un númerito. Este número se conoce como "Atributo de color" y nos va a servir para especificar los colores mediante la siguiente instrucción:

```
COLOR primerPlano, fondo
```

Por ejemplo, si queremos escribir con letras amarillas sobre fondo azul (muy típico), no tenemos más que usar una instrucción

```
COLOR 14, 1
```

antes de las correspondientes instrucciones PRINT o INPUT.

Al usar una instrucción COLOR, se cambian los colores de la pantalla para sucesivas instrucciones de escritura en pantalla hasta que se vuelva a cambiar usando otra instrucción COLOR. No hace falta poner una instrucción COLOR delante de cada PRINT o INPUT si no vamos a cambiar el color.

Como color de primer plano podemos usar cualquiera de los 16 atributos de color disponibles. Como color de fondo solo los 8 primeros. Si no especificamos color de fondo, se conservará el que hubiera.

El color de fondo en principio sólo afecta al trozo de pantalla que hay por detrás de los caracteres que escribimos. Si queremos colorear toda la pantalla de un determinado color, una forma muy rápida de hacerlo es especificar una instrucción COLOR justo antes de una CLS, por ejemplo:

```
COLOR 15, 4
CLS
```

Hará que toda la pantalla se coloree de rojo oscuro. En sucesivas instrucciones PRINT se escribirá lo que sea en color blanco fuerte sobre este rojo mientras no usemos otra instrucción COLOR. Hay que tener cuidado de no usar el mismo color para primer plano y para fondo. Si lo hacemos no pasa nada, nuestro programa va a seguir funcionando perfectamente, pero sería como escribir con lápiz blanco en un papel, no se vería nada de nada.

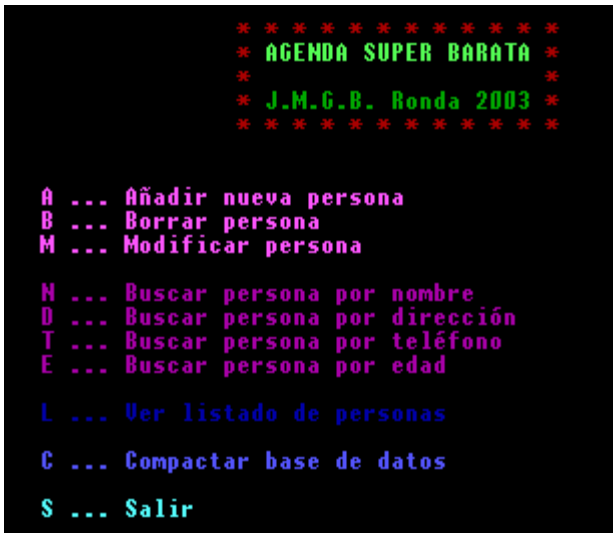
También podemos conseguir que los caracteres aparezcan en la pantalla parpadeando. Para hacerlo no hay más que sumar 16 al color de primer plano que queramos, por ejemplo:

```
COLOR 30, 2
```

Hará que el texto escrito posteriormente aparezca de amarillo parpadeando sobre fondo verde. (14 del amarillo más 16 son 30). No se puede hacer que el fondo parpadee.
Una cosa importante a tener en cuenta con el parpadeo es que NO VA A FUNCIONAR si estamos ejecutando QBasic desde Windows en una ventana. Para que funcione habrá que pasar a pantalla completa pulsando ALT+ENTER o el botón correspondiente de la barra de botones de la ventana, y aún así a la primera tampoco funciona algunas veces, lo que se hace es afectar al color de fondo.

No es conveniente abusar del parpadeo, puede resultar molesto para la vista. Se debe de utilizar sólo para resaltar pequeños mensajes o símbolitos en la pantalla.

Como ejemplo veamos como podría quedar una portada sencilla en colores para un programa de agenda que integre todo lo que vimos en el tema de ficheros.



Para conseguir esto no habría mas que insertar las instrucciones COLOR correspondientes en los lugares adecuados, como se ve aquí:

```
CLS
PRINT
COLOR 4
PRINT , , "* * * * * * * * * * * *"
PRINT , , "* ";
COLOR 10: PRINT "AGENDA SUPER BARATA ";
COLOR 4: PRINT "*"
PRINT , , "*                     *"
PRINT , , "*";
COLOR 2: PRINT " J.M.G.B. Ronda 2003 ";
COLOR 4: PRINT "*"
PRINT , , "* * * * * * * * * * * *"
PRINT
PRINT
COLOR 13
PRINT , "A ... Añadir nueva persona"
PRINT , "B ... Borrar persona"
PRINT , "M ... Modificar persona"
PRINT
COLOR 5
PRINT , "N ... Buscar persona por nombre"
PRINT , "D ... Buscar persona por dirección"
PRINT , "T ... Buscar persona por teléfono"
PRINT , "E ... Buscar persona por edad"
PRINT
COLOR 1
PRINT , "L ... Ver listado de personas"
PRINT
COLOR 9
PRINT , "C ... Compactar base de datos"
PRINT
COLOR 11
PRINT , "S ... Salir"
```

Estas son las instrucciones sólo para dibujar la pantalla. El resto para que funcione el menú habría que progrmarlo por ejemplo con la función INKEY$ y

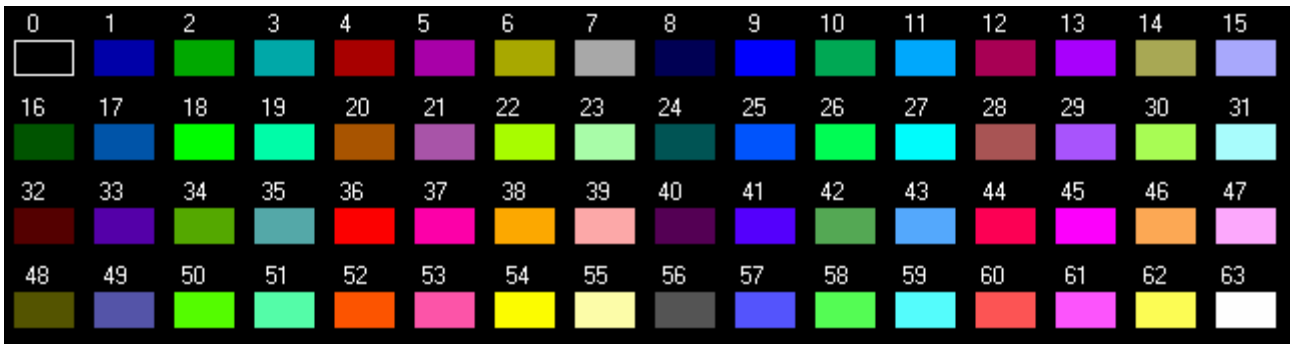
un SELECT CASE que según la tecla pulsada llame a cada uno de los módulos del programa.

2.4.4 - REDEFINIR COLORES

En el apartado anterior hemos visto cómo conseguir que nuestros programas escriban en la pantalla usando distintos colores. Esto, junto con los semigráficos que veremos en el apartado siguiente, es más que suficiente para hacer que nuestros programas tengan una interfaz en modo texto bastante conseguida.

Observando la paleta VGA de 16 colores se puede comprobar que los colores están un poco "descoloridos". No es que al monitor le pase nada cuando entramos a MS-DOS, es que son los que hay y están definidos así. También nos puede pasar que queramos usar determinados colores, por ejemplo para construir un logotipo, y no nos venga bien ninguno de los que tenemos.

Esto se puede arreglar un poco. El modo de pantalla de texto nos permite usar 16 colores SIMULTÁNEAMENTE, pero estos no tienen porqué ser siempre los mismos. Los podemos cambiar fácilmente por cualquier otro de esta paleta más amplia de 64 colores.



Podemos intercambiar cualquiera de los 16 atributos de color que tenemos disponibles por uno de estos 64 colores usando la siguiente instrucción:

```
PALETTE original, nuevo
```

Vamos a ver lo que significa esto. Puede dar lugar a confusiones:

- Original es el ATRIBUTO de color que queremos cambiar (Un número entre 0 y 15)
- Nuevo es el color de esta paleta que queremos usar (Un número entre 0 y 63)

Supongamos que queremos cambiar el atributo 3 (Que originalmente es un azul grisaceo muy feo) por un celeste más bonito que aparece en nuestra paleta con el índice 43. No habría más que hacer:

```
PALETTE 3, 43
```

Y a partir de ahora cada vez que usemos la instrucción

```
COLOR 3
```

se escribirá con nuestro celeste. El 3 ya no es el azul grisaceo feo.

Podemos intercambiar los colores todas las veces que queramos e incluso asignar valores repetidos a distintos atributos. Lo que hay que tener en cuenta es que sólo vamos a ver en la pantalla 16 colores a la vez, pero estos podrán ser cualquiera de los 64.

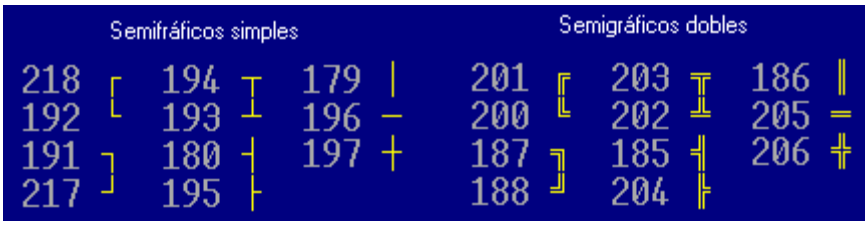
Otra cosa que hay que tener en cuenta es que los cambios hechos con la instrucción PALETTE también afectan a todo lo que ya haya escrito en la pantalla del color que hemos cambiado. Imagina que hay escrito algo en color 10 original (Verde fuerte), y ahora asignamos al atributo 10 el color de paleta 51 para seguir escribiendo en verde más suave. Automáticamente todo lo escrito en verde fuerte pasa a ser verde suave. Para poder ver los dos verdes a la vez habría que usar otro atributo y dejar el 10 como estaba. Con un poco de práctica haciendo esto se puede conseguir que determinados textos se enciendan o se apaguen poco a poco (Cambiando los colores por otros cada vez más oscuros o más claros dentro de un bucle), algunos efectos interesantes como ocultar el rótulo "Presione cualquier tecla y continúe".

La instrucción PALETTE no tiene ningún interés para aprender a programar, pero nos puede servir para dar a nuestros programas un toque personal o extraño al utilizar combinaciones de colores que se salen de los 16 tan típicos de la amplia mayoría de programas de MS-DOS. Hay que recordar que como fondo sólo podemos usar los ocho atributos primeros (0 a 7), pero ya podemos conseguir cosas como escribir sobre fondo amarillo o blanco. Pero con cuidado!, que no haya que usar gafas de sol delante del ordenador.

2.4.5 - CARACTERES SEMIGRÁFICOS

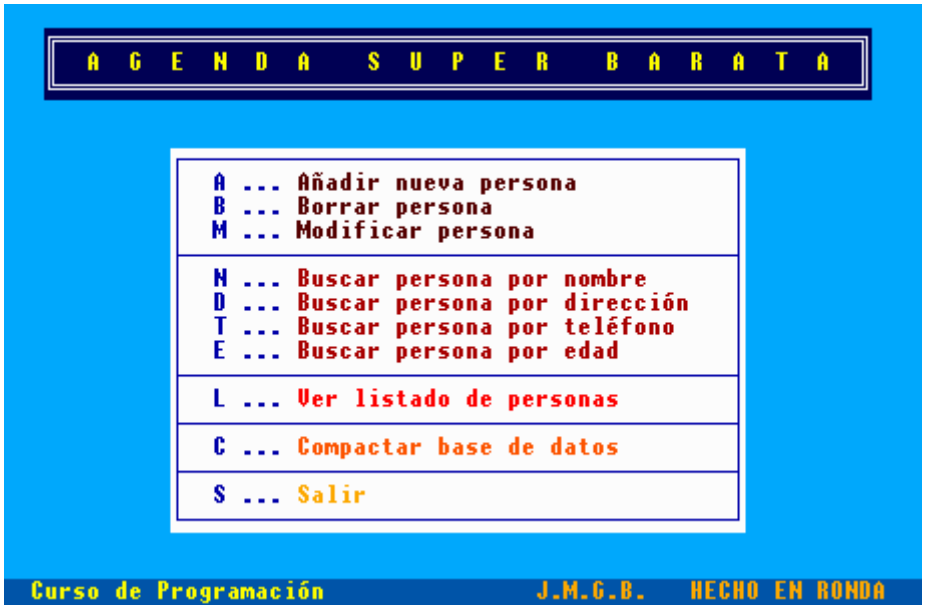
Ya sabemos escribir en colores. Con un poco de idea ya podemos dibujar recuadros aplicando distintos colores de fondo a distintas zonas de la pantalla, pero todavía podemos llegar un poco más allá. Podemos dibujar recuadros con bordes y líneas horizontales, verticales y esquinas usando unos caracteres especiales que nos permiten "ensamblar" las líneas en la pantalla como si se tratara de un puzzle: Los caracteres SEMIGRÁFICOS, casi gráficos.

Estos caracteres no los tenemos en el teclado, y por lo tanto para conseguirlos habrá que recurrir a sus códigos ASCII, que son los siguientes:



Recordemos que para introducir un código ASCII hay que pulsar simultáneamente la tecla alternativa (ALT) y el código correspondiente en el bloque numérico situado a la derecha del teclado.

Para demostrar la utilidad de los caracteres semigráficos vamos a crear una nueva portada para nuestro programa de agenda. Va a ser esta, que usa también colores personalizados.



Como se puede ver, bastante conseguida. Ya no tiene casi nada que envidiarle a la interfaz de un programa comercial de MS-DOS. Lo que hace falta es que todo lo demás también funcione perfectamente y sea útil.

Ahora vamos con el listado del programa necesario para dibujar esto:

```
PALETTE 0, 32
PALETTE 3, 11
PALETTE 4, 8
PALETTE 5, 17
PALETTE 7, 63
PALETTE 10, 4
PALETTE 11, 52
PALETTE 12, 36
PALETTE 13, 38
PALETTE 14, 54
COLOR 15, 3
CLS
COLOR 15, 4
LOCATE 2, 11: PRINT ""; STRING$(57, 205); ""
LOCATE 3, 11: PRINT "|"; SPACE$(57); "|"
```

```
LOCATE 4, 11: PRINT ""; STRING$(57, 205); ""
COLOR 14, 4: LOCATE 3, 14
PRINT "A  G  E  N  D  A    S  U  P  E  R    B  A  R  A  T  A"
COLOR 1, 7
LOCATE 7, 20:  PRINT "---------------------------------------"
LOCATE 8, 20:  PRINT "|  A ...                               |"
LOCATE 9, 20:  PRINT "|  B ...                               |"
LOCATE 10, 20: PRINT "|  M ...                               |"
LOCATE 11, 20: PRINT "---------------------------------------"
LOCATE 12, 20: PRINT "|  N ...                               |"
LOCATE 13, 20: PRINT "|  D ...                               |"
LOCATE 14, 20: PRINT "|  T ...                               |"
LOCATE 15, 20: PRINT "|  E ...                               |"
LOCATE 16, 20: PRINT "---------------------------------------"
LOCATE 17, 20: PRINT "|  L ...                               |"
LOCATE 18, 20: PRINT "---------------------------------------"
LOCATE 19, 20: PRINT "|  C ...                               |"
LOCATE 20, 20: PRINT "---------------------------------------"
LOCATE 21, 20: PRINT "|  S ...                               |"
LOCATE 22, 20: PRINT "---------------------------------------"
COLOR 0, 7
LOCATE 8, 29:   PRINT "Añadir nueva persona"
LOCATE 9, 29:   PRINT "Borrar persona"
LOCATE 10, 29: PRINT "Modificar persona"
COLOR 10
LOCATE 12, 29: PRINT "Buscar persona por nombre"
LOCATE 13, 29: PRINT "Buscar persona por dirección"
LOCATE 14, 29: PRINT "Buscar persona por teléfono"
LOCATE 15, 29: PRINT "Buscar persona por edad"
COLOR 12: LOCATE 17, 29: PRINT "Ver listado de personas"
COLOR 11: LOCATE 19, 29: PRINT "Compactar base de datos"
COLOR 13: LOCATE 21, 29: PRINT "Salir"
COLOR 14, 5: LOCATE 25, 1: PRINT SPACE$(80);
LOCATE 25, 10: PRINT "Curso de Programaci¢n";
COLOR 13: LOCATE 25, 46: PRINT "J.M.G.B.   HECHO EN RONDA";
SLEEP
```

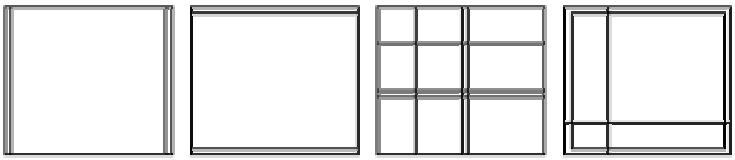
¿A que asusta? Como se puede ver, ya es un listado bastante largo.

Antes de seguir hay que aclarar un pequeño problema que tiene: En Windows no podemos representar los caracteres semigráficos y los he sustituido por guiones, tuberías y escuadras para tener una idea de dónde va cada cosa. Al copiar este listado a QBasic tendrás que sustituirlos por los correspondientes semigráficos usando la tabla de códigos ASCII. Los del recuadro superior son de línea doble y los de abajo sencillos.

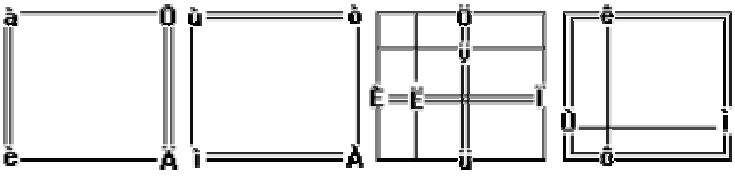
Para empezar a desliar este listado de código tan largo hay que tener en cuenta que es un esquema secuencial. No hay bucles ni operaciones complejas. Así ya se ve bastante más sencillo. Si te lías con los colores redefinidos quita las instrucciones PALETTE y usa solo los colores normales. Así de paso te vas aprendiendo sus 16 códigos, también te servirán para algunos otros lenguajes de programación. Cuando esté terminado redefine los colores que quieras.

Además de los caracteres semigráficos sencillos y dobles que hemos visto aquí, existen otros conocidos como "mixtos" que se pueden usar en las esquinas y en las uniones de líneas horizontales y verticales para ensamblar líneas sencillas y

dobles, es lo que se ve en las esquinas y uniones de las líneas que forman este dibujo.



Esto podría resultar interesante, lo que ocurre es que estos caracteres semigráficos mixtos no son muy comunes y no aparecen en todas las tablas de caracteres ASCII. En el caso de que el usuario de nuestro programa esté utilizando una de estas tablas de caracteres extraña, no podrá ver los caracteres estos y nuestro diseño de pantalla quedará roto. Normalmente las esquinas de los recuadros se cambiarán por vocales acentuadas y otros símbolos extraños, algo así como esto:



Como no nos merece la pena correr el riesgo de que nuestros diseños de pantalla queden así de estropeados, lo mejor es evitar el uso de los semigráficos mixtos. Aunque los veamos en nuestra tabla de caracteres ASCII, puede que el usuario del programa no los tenga y no pueda verlo correctamente. Lo mismo ocurre al imprimir si la impresora utiliza una tabla de caracteres ASCII distinta. Recordar que estamos en MS-DOS y no hay fuentes True Type ni nada parecido. Con un poco de imaginación se pueden construir diseños de pantallas más que aceptables usando sólo los semigráficos simples y dobles que ya hemos visto junto con los caracteres de relleno que veremos en el apartado siguiente.

Para terminar con los semigráficos vamos a ver un pequeño algoritmo que usado como un procedimiento nos sirva para dibujar un recuadro en las posiciones de pantalla que nos interesen, algo muy útil.

```
LOCATE v, h: PRINT "╔"; STRING$(largo - 2, "="); "╗";
FOR vv = v + 1 TO v + alto - 1
   LOCATE vv, h: PRINT "║"; SPACE$(largo - 2); "║";
NEXT
LOCATE v + alto - 1, h: PRINT "╚"; STRING$(largo - 2, "="); "╝";
```

Lo que hace es sencillo. Dibuja un cuadro con el borde de línea doble con la esquina superior derecha en la posición v,h y la anchura y la altura la obtiene de las variables largo y alto. La primera línea dibuja la parte superior, el bucle FOR dibuja la zona intermedia rellenando lo de dentro con espacios, y la última línea dibuja el borde inferior. Ponemos un punto y coma al final de cada línea para que si dibujamos el recuadro en la parte baja de la pantalla no se produzca el desplazamiento automático y se nos estropee todo el diseño de pantalla.

Este ejemplo se podría ampliar especificando los colores en la llamada a la función o escribiendo algún tipo de barra de título como si fuera una ventana de las del editor de QBasic.

2.4.6 - CARACTERES ESPECIALES DE RELLENO

Además de los semigráficos también existen otros caracteres que nos van a ser muy útiles para de diseñar las pantallas de nuestros programas en modo texto. Son los caracteres de relleno, aquí están sus códigos ASCII, porque tampoco los tenemos en el teclado:

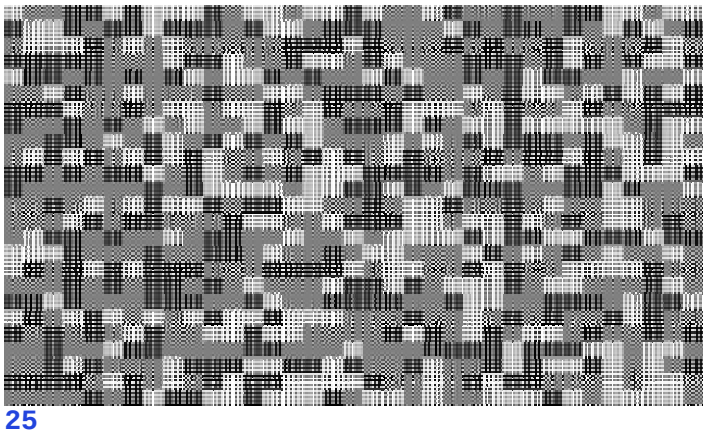
Estos caracteres nos van a permitir rellenar zonas enteras de la pantalla para evitar que se vean todo del mismo color. Los más utilizados son los tres primeros que nos ofrecen distinta densidad de puntos. El cuarto rellena todo con el color de primer plano, y los dos últimos se usan sólo para cosas como dar efectos de sombras a recuadros y poco más.

Para usar estos caracteres se suele recurrir a un bucle FOR que recorre la pantalla por filas y dentro dibuja series del carácter correspondiente usando la función STRING$.
Vamos con un ejemplo que va a rellenar con el carácter 177 la mitad izquierda de la pantalla.

```
FOR v = 1 TO 25
       LOCATE v,1: PRINT STRING(40,177);
NEXT
```

Una cosa que aparece al usar grandes cantidades de estos caracteres es el llamado "efecto Moiré" (Muaré), consistente en una serie de lineas curvas que aparecen dibujadas en la pantalla. Se produce por una interferencia entre los puntos que forman estos caracteres de relleno y los píxeles físicos del monitor. Si nuestro monitor nos da ese problema, no se puede evitar, pero sí atenuar usando colores de primer plano y de fondo que no sean demasiado distintos.

Ahora vamos con otro diseño más elaborado que nos ofrece un diseño de fondo de este tipo:



```
FOR v = 1 TO 25
        FOR h = 1 TO 79 STEP 2
                LOCATE v, h: PRINT STRING$(2, INT(RND * 3) + 176);
        NEXT
NEXT
```

Aquí se elige aleatoriamente uno de los tres caracteres disponibles en el momento de dibujarlo. Se han agrupado los caracteres de dos en dos para que salgan cuadros más o menos cuadrados. El siguiente ejemplo es parecido, pero esta vez los cuadros aparecerán ordenados ya que se usa un contador en vez de los números aleatorios.

```
c = 0
FOR v = 1 TO 25
        FOR h = 1 TO 79 STEP 2
                c = c + 1
                IF c = 3 THEN c = 0
                LOCATE v, h: PRINT STRING$(2, 176 + c);
        NEXT
NEXT
```

El uso de estos caracteres de relleno junto con los recuadros de colores hechos a base de semigráficos es una buena forma de dar a nuestros programas en modo texto una apariencia bastante aceptable.

2.4.7 - CARACTERES NO IMPRIMIBLES

Si se observa la tabla de códigos ASCII se puede ver que los caracteres que van desde el 0 hasta el 31(El 32 es el espacio en blanco) son símbolos extraños y además no se pueden escribir usando la tecla ALT.

Estos caracteres están asociados a teclas del teclado como Enter, escape, tabulador o retroceso entre otras y a ordenes especiales para el ordenador y la impresora como son saltos de página y de línea, fin de fichero, toque de timbre, etc.

Estos caracteres no están pensados para ser dibujados en la pantalla ni para ser impresos. Tienen asociados esos símbolos porque alguno tenían que tener,

pero ya que los tenemos ahí puede ser que en algunos casos queramos escribirlos en la pantalla. Vamos con sus códigos ASCII...

Y ahora vamos a ver como escribirlos. (Solo en QBasic y en los programas de MS-DOS. En Windows no se puede)

Hay dos formas. La primera sería usar la función CHR$, por ejemplo para dibujar un corazón bastaría con poner PRINT CHR$(4). La segunda, más cómoda es pulsar la combinación de teclas CONTROL + P y después, a continuación ALTERNATIVA + el código en el teclado numérico, por ejemplo para dibujar el corazón habría que pulsar CTRL+P y ALT+4 con lo que este símbolo quedaría directamente dibujado en la pantalla.

Esto de CTRL+P nos vale para dibujar estos caracteres especiales en la mayoría de programas de MS-DOS así como en el propio intérprete de comandos (Símbolo C:\>).

Hasta aquí muy bien. Ya podemos dibujar caras, flechas y otros símbolos en nuestras pantallas, pero no podemos olvidar que estos caracteres son especiales y hay que tener unas ciertas precauciones. La primera es que si un símbolo nos da algun problema como que se nos descoloquen los demás, pues directamente evitar usarlo y nada más. Otras precaucione son no usar los símbolos que no llevan asociado ningún carácter como son el de código ASCII 0 y tener en cuenta que cada vez que nuestro programa dibuje un ASCII 7 se oirá un pitido en el altavoz interno o bien sonará el sonido predeterminado de Windows.

Estos caracteres los usaremos exclusivamente en instrucciones PRINT y no deberemos almacenarlos en ficheros, especialmente en ficheros secuenciales . Si los intentaremos imprimir van a provocar fallos en la impresora. De hecho si intentas imprimir tal cual está la tabla de caracteres ASCII que viene en la ayuda de QBasic y que incluyes estos símbolos observarás cosas como alguna línea se rompe o incluso que se hace un salto de página y un trozo de la tabla se imprime en la siguiente hoja. Esto se debe a que se han encontrado caracteres de saltos de línea y de página que la impresora ha interpretado como lo que son.

También hay que tener precaución si estos caracteres aparecen de forma literal en las instrucciones del código fuente del programa. Si imprimes dicho código puedes encontrarte con problemas similares cada vez que salga uno de estos caracteres. Por lo tanto si piensas imprimir el código fuente del programa utiliza funciones CHR$ en lugar de los caracteres literales. También hay que saber que estos caracteres no se pueden ver en Windows, por lo que si abres el fichero BAS del código de tu programa en un editor de Windows como por ejemplo el

bloc de notas, directamente no los verás o los verás marcados como un cuadro negro.

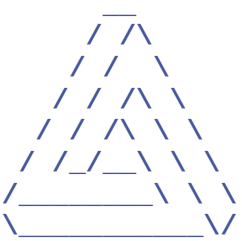
2.4.8 - ESMÁILIS y ASCII-ART

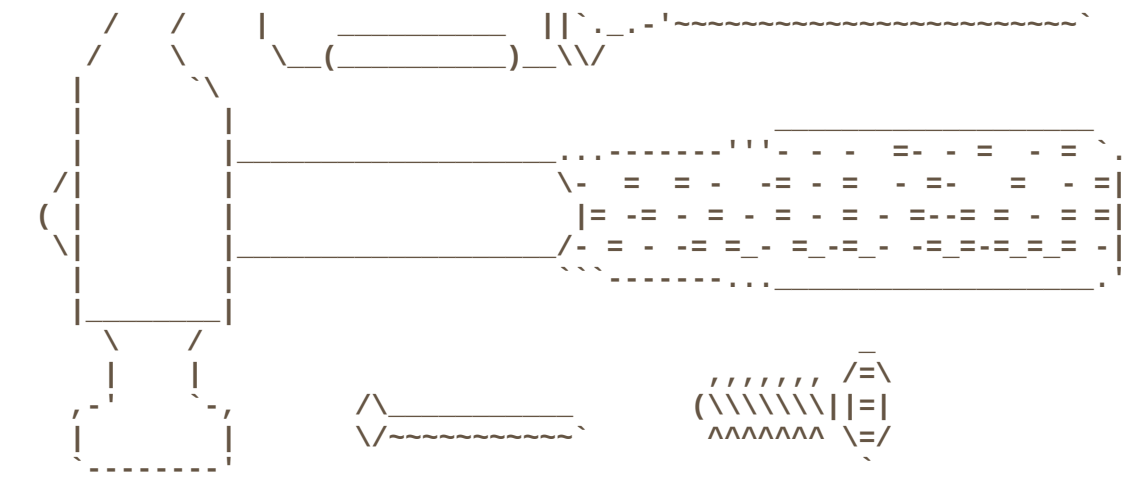
Para enriquecer un poco más las pantallas de nuestros programas en modo texto podemos usar la forma que tienen los propios caracteres para hacer pequeños dibujos.

Lo más sencillo de esto es lo que se conoce como esmáilis (Correctamente se escribe smiley) y que estamos acostumbrados a utilizar en los foros y chats de Internet. Algunos de los más comunes y de significado conocido son los siguientes:

:-) :-(:-D :-o X-D

Se pueden encontrar listados muy extensos llenos de estos símbolitos junto con su significado, pero de todas formas puede que estos esmáilis sean muy poca cosa para decorar nuestros programas. Por eso podemos recurrir a diseños más complicados conocidos como ASCII-ART. Vamos con unos ejemplos...





Podemos encontrar montones de ellos o atrevernos nosotros mismos a dibujar los nuestros (Por ejemplo un logotipo).

Desde el punto de vista de la programación lo único que hay que tener en cuenta es que hay que dibujarlos línea por línea usando una instrucción PRINT para cada una, y que en muchos casos no podemos olvidar incluir espacios a la izquierda de determinadas líneas para que el dibujo no se estropee. Por ejemplo, para dibujar la primera de las anteriores figuras habría que hacer...

```
PRINT "       __"
PRINT "     / /\"
PRINT "    / /  \"
PRINT "   / / /\ \"
PRINT "  / / /\ \ \"
PRINT " / /_/__\ \ \"
PRINT "/________\ \ \"
PRINT "\__________\/"
```

Estos dibujos fueron concebidos inicialmente para ser impresos en impresoras antiguas que no soportaban la impresión de gráficos, por lo que en principio es suficiente con que sean en blanco y negro, pero también nos podemos molestar en ponerlos de varios colores dibujándolos con distintas instrucciones PRINT.

También podemos construir grandes letras para dibujar los títulos y rótulos de nuestros programas, por ejemplo...

```
|_|  \__\_\_|_\|___/ |_|  \___/  \_/    \_/\_/  /_/\_\  |_| /___|
```

Estas letras están construidas usando principalmente barras y símbolos de subrayado. También es común construirlas usando los caracteres semigráficos o el símbolo # repetido (Esto último es lo que hace la instrucción banner de Linux).

Para dibujar un rótulo con las letras anteriores podemos hacerlo de forma literal con cuatro instrucciones PRINT (Una por cada línea) y los trozos de letras entre comillas...

```
PRINT " _  _  ___  ___  _  _  ___     ___  _  _    ___  ___  _  _
___    _"
PRINT "| || || __|/ __|| || |/ _ \   | __|| \| |  | _ \/ _ \| \| ||
\  /_\"
PRINT "| __ || _|| (__|| __ | (_) |  | _||| .` |  |   / (_) | .` ||
|) |/ _ \"
PRINT "|_||_||___|\___||_||_|\___/   |___||_|\_|
\_|_\\___/|_|\_||___//_/ \_\"
```

Pero ya que estamos programando, lo más cómodo sería tener un procedimiento que nos dibujara el rótulo que queramos en la posición que le digamos solo con hacer una llamada del tipo...

```
DibujaRotulo "Hecho en Ronda", 5, 10
```

Habría que construir un programa con muchas instrucciones Mid$ que vaya construyendo las letras línea por línea. Se puede encontrar en la sección de programas terminados de esta web.