Mallard BASIC

Para

AMSTRAD

SPEN,SA

Mallard BASIC

Para

AMSTRAD

SPEN,SA

Autor:

SPEN,SA

Centro de Informática Educativa Aplicada a la Enseñanza

Editor:

TASOFT

INFANTA Mª TERESA,23
TF: (91) 259 50 54 - Madrid

INDICE

1. INTRODUCCION AL SISTEMA OPERATIVO CP/M.

Un Sistema Operativo es un conjunto de programas y utilidades que el fabricante del ordenador suministra junto con la máquina. El Sistema Operativo que utiliza el ordenador AMSTRAD PCW 8256 es CP/M PLUS. El Sistema Operativo CP/M es el más extendido entre ordenadores de ocho bits, lo que proporciona una amplia biblioteca de programas compatibles a los usuarios de ordenadores que tengan este Sistema Operativo.

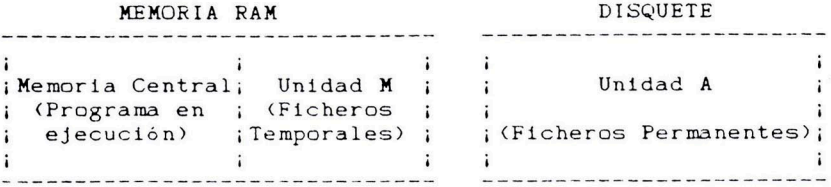
CP/M PLUS proporciona una colección de recursos para la gestión de la información grabada en los discos que está estructurada en ficheros. Mediante ciertos comandos, que se estudiarán a lo largo de este capitulo, se pueden borrar ficheros, copiarlos, acceder a ellos, etc.

1.1.-Configuración del Sistema.

Antes de ver los comandos de gestión de discos es preciso conocer la estructuración que el ordenador hace del espacio de almacenamiento.

El PCW 8256 divide la memoria en dos partes: Una, actúa como memoria central propiamente dicha, guardando el programa y los datos que son objeto de proceso. La otra parte de la memoria, llamada *unidad M* simula un disco interno cuyo funcionamiento es análogo al disquete que se introduce en la unidad de disco con la particularidad de que la información almacenada en la unidad M se borra cuando se desconecta el aparato. Recordemos que aunque la unidad M actúe como si fuera un disco, físicamente es una porción de memoria RAM, memoria volátil. Por las características reseñadas, la unidad M es llamada *disco virtual*.

La unidad de disco externa, donde introducimos el disquete, se llama *unidad A*.



En el esquema anterior se observa el particionamiento del espacio físico de almacenamiento. En la unidad A quedarán

los ficheros de una forma permanente a menos que se de una orden de borrarlo. La unidad M es utilizable para guardar en ella temporalmente ficheros. Es de notar que por ser la unidad M una unidad interna, los accesos de disco a memoria y de memoria a disco son más rápidos cuando se trabaja con la unidad M que cuando se trabaja con la unidad A.

1.2. Ordenes permanentes y transitorias.

El conjunto de programas que componen el Sistema Operativo CP/M se encuentra en el disco de CP/M. (Disco rosa, cara 2).

El drive debe contener un disco del sistema. Esta orden carga en memoria una serie de utilidades de CP/M que son directamente utilizables. Por ello se llaman órdenes residentes. Otras, por el contrario, están en el disco de CP/M, pero no en memoria y para su ejecución es preciso que dicho disco se encuentre en el drive. A estas órdenes almacenadas en disco pero no permanentemente en memoria se las llama órdenes transitorias.

1.3. Localización de un fichero. Unidad Implícita.

Como se ha expuesto anteriormente, la información almacenada en cualquiera de los discos (A y M) está estructurada en forma de ficheros. Desde este punto de vista podemos pensar en los ficheros como unidades de información con las cuales trabaja el Sistema Operativo. De esta manera, mediante un comando del Sistema Operativo se puede borrar o copiar un fichero pero no se puede realizar este tipo de operaciones con medio fichero o con un trozo de fichero. En CP/M los ficheros tienen un nombre de un máximo de ocho caracteres y una extensión que indica el contenido del fichero.

Ejemplos:

```
NOMINA.BAS   (Indicaría un fichero en BASIC)
PIP   .COM   (Indicaría un fichero comando)
DATOS .      (Indicaría un fichero de datos)
|     | | |
------- ---
   ↓      |----> Extensión
Nombre
```

Para localizar un fichero es preciso indicar, además del nombre y de la extensión, la unidad donde se aloja (A o M) seguido de dos puntos (:).

Ejemplos:

```
A:NOMINA.BAS   (Indicaría un fichero en BASIC en A)
M:PIP    .COM  (Indicaría un fichero  comando en M)
M:DATOS  .     (Indicaría un fichero de datos en M)
```

De esta manera el Sistema Operativo sabe en qué disco ha de buscar un determinado fichero.

Cuando el Sistema Operativo espera una orden, presenta la unidad de disco sobre lo que está trabajando implícitamente (A ó M) seguida del signo ">". Esto quiere decir que si no aparece el nombre de la unidad donde se encuentra el fichero se buscará en la unidad implicita.

Cuando se carga por primera vez el Sistema Operativo la unidad implícita es la A y en pantalla aparece:

Si se quiere cambiar la unidad implícita a la unidad M se escribirá:

Y después de pulsar <RETURN> aparecerá:

```
A>M:
M>
```

Lo que nos indica que la nueva unidad implícita es la M.
Del mismo modo, para que sea la unidad implícita la unidad escribiremos:

```
A>M:
M>A:
```

Después de escribir A: y de pulsar <RETURN> volverá a ser la unidad A la unidad implícita.

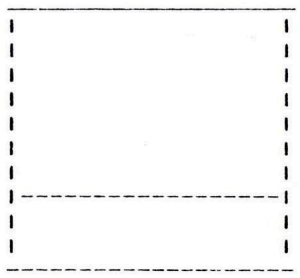
```
A>M:
M>A:
A>
```

1.4. COMO INICIALIZAR UN DISCO.

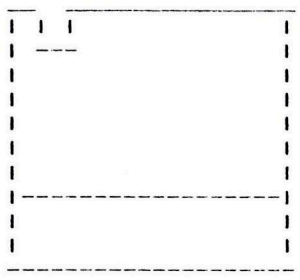
Inicializar o formatear un disco es la operación que hay que realizar para que se puedan grabar datos en él. Se pueden formatear discos vírgenes o discos que ya contengan

información, pero si se formatea uno de estos discos ya grabados, se pierden todos los ficheros que contuviera.

Para formatear un disco es preciso que éste no se encuentre protegido contra escritura. Para ello debe tener sacada la patilla según se muestra en el dibujo.



Disco libre para ser formateado



Disco protegido contra escritura.

Una vez que tenemos el disco preparado para ser formateado introducimos el disco de CP/M en el drive y se escribe:

DISCKIT

Ante esta orden (transitoria) el Sistema Operativo nos ofrecerá un menú de opciones. Una de ellas es la opción "Inicializar" que es la que elegiremos después de introducir en el drive el disco que queremos formatear. Posteriormente iremos obedeciendo las órdenes que nos dé el programa.

1.5. Directorios.

Se llama directorio de un disco a un área especial de éste que contiene información de cada fichero. Si queremos que aparezcan en pantalla los nombres de los ficheros que tenemos en un disco escribiremos la orden DIR (orden permanente) seguida de la unidad.

Ejemplos:

```
DIR A:
```

En pantalla aparecerá el nombre de cada uno de los ficheros grabados en la unidad A.

```
DIR M:
```

Ahora aparecerán los ficheros grabados en la unidad M.

```
DIR
```

Con la instrucción DIR sin parámetros aparecen los ficheros grabados en la unidad implícita.

1.6. Cómo copiar un disco.

Para copiar un disco se utiliza el programa diskit (transitorio), eligiendo la opción "Copiar" después de insertar el disco que se desea copiar. El programa lo irá leyendo por partes y a medida que concluya de leer cada parte, solicitará que se introduzca el disco "copia". Este disco no debe estar protegido contra escritura. Cuando se introduce el disco empieza a grabar sobre él lo que ha leído del disco original. Una vez que termina de grabar cada una de las partes solicita que se introduzca de nuevo el disco original. Este proceso se repite hasta que termina de grabar completamente el disco.

1.7. Cómo copiar un fichero.

Para copiar un fichero se utiliza la orden PIP (transitoria). Cuando se teclea la orden PIP (programa de intercambio de periféricos) se carga este programa y el ordenador responde presentando en pantalla un asterisco. En este momento el programa PIP está cargado y se puede sacar el disco de CP/M del drive.

Imaginemos que se quiere copiar un fichero llamado "original.bas", que tenemos grabado en un disco, en un segundo disco y se desea que aparezca con el nombre "copia.bas". Se operará de la forma siguiente:

```
A>PIP  (Con el disco de CP/M en el drive)

*M:copia.bas=A:original.bas (Con el disco del
              fichero original en el drive).

*A:copia.bas=M:copia.bas (Con el disco donde se
              quiere hacer la copia introducido
              en el drive.
```

El fichero que va antes del signo igual, es el fichero destino, donde se quiere hacer la copia. El fichero que va detrás del signo igual, es el fichero orígen, que se quiere copiar. El proceso ha consistido en grabar el fichero en la unidad M y desde la unidad M se ha grabado en el disco donde se deseaba hacer la copia.

1.8. Cómo borrar un fichero

Para borrar un fichero se utiliza el comando ERA ó ERASE (residente). Su formato es:

ERA <nombre del fichero. extensión>

Supongamos que se quiere borrar un fichero llamado "NOVALE.BAS". Se escribirá:

A>ERA NOVALE.BAS

El disco no debe estar protegido contra escritura.

1.9. Cómo cambiar de nombre un fichero.

Para cambiar de nombre un fichero se utiliza el comando REN ó RENAME (residente). Su formato es:

REN <nombre nuevo>=<nombre antiguo>

Pongamos el ejemplo de que se quiere cambiar el nombre de un fichero llamado "ayer.bas" para pasar a llamarse "hoy.bas". Se deberá escribir:

REN hoy.bas=ayer.bas

1.10. Cómo listar un fichero.

Se puede listar el contenido de un fichero en ASCII mediante el comando TYPE (residente). El formato es:

TYPE <nombre del fichero>

Para listar el contenido de un fichero llamado "datos", se procede de la siguiente forma:

TYPE datos

Es preciso no olvidar la extensión al dar el nombre del fichero.

2. ELEMENTOS DEL LENGUAJE BASIC.

Podemos definir un lenguaje de programacion como un conjunto de instrucciones y reglas sintácticas que permiten la comunicación hombre-ordenador.

Las instrucciones a su vez operan con datos. Los datos, en Basic, pueden venir dados en forma de constantes y variables.

2.1. CONSTANTES

Las constantes pueden ser de dos tipos:

* CONSTANTES NUMERICAS

Las constantes numéricas pueden ser de dos tipos: numéricas y alfanuméricas. Las constantes numéricas son números que no varían a lo largo del programa. Así, por ejemplo, un 3 siempre vale 3, por eso decimos que es una constante y como tiene valor numérico decimos que es una constante numérica. Estas constantes, como números que son, permiten que realicemos operaciones aritméticas con ellas. Las constantes numéricas son números cuyos dígitos están comprendidos del *0* al *9*.

Ejemplos de constantes numéricas:

3 5 6.34 789 4156 321

* CONSTANTES ALFANUMERICAS

Las constantes alfanuméricas son un conjunto de caracteres que no varían a lo largo del programa. Pueden ser letras, números y caracteres especiales, o una combinación de letras y/o números y/o caracteres especiales. Estas constantes se encierran siempre entre comillas.

Ejemplos de Constantes Alfanuméricas:

"Madrid"

"7 de Julio San Fermín"

"E.G.B."

Si los caracteres de una constante alfanumérica son números, no se interpretan como cantidad numérica, sino como un conjunto de caracteres con los cuales no se pueden realizar operaciones aritméticas. A veces es útil que un conjunto de números sean una constante alfanumérica en vez de numérica. Por ejemplo, el número de un teléfono, aunque esté compuesto de una serie de caracteres numéricos, no suele ser utilizado para realizar con él sumas, restas, multiplicaciones,etc. (Si sumamos dos números de teléfonos tendremos un tercer número que no nos servirá para nada). Lo lógico es que el número de un teléfono sea una constante alfanumérica en vez de una numérica. Para ello encerramos la constante entre comillas y de esta manera se considerará como constante alfanumérica.

2034567 ---------- CONSTANTE NUMERICA

"2034567" ---------- CONSTANTE ALFANUMERICA

El primero de los números anteriores es una constante numérica, por tanto, es susceptible de realizar con él operaciones aritméticas. En cambio, el segundo no es posible operarlo como un número.

Las constantes alfanuméricas permiten incluir espacios en blanco, de modo que podemos poner "203 45 67".

Las constante numéricas, como números que son, no permiten espacios entre sus dígitos.

2.2. Variables.

Se puede definir la variable como *un espacio de memoria al que se le da un nombre y que contiene una información que puede variar a lo largo de la ejecución de un programa*. Esta información, como en el caso de las constantes, puede ser numérica o alfanumérica.

Se llaman variables porque EL VALOR QUE CONTIENEN se puede modificar a lo largo del programa, pero SU NOMBRE SIGUE SIENDO EL MISMO.

* VARIABLES NUMERICAS

Las variables numéricas contienen un número. Por tanto se pueden realizar cálculos aritméticos con ellas. Se definen mediante el empleo de letras y números siguiendo las reglas que se dan a continuación:

1. El nombre de una variable tiene que empezar por una letra.

2. No puede contener blancos ni signos distintos de letras, pero puede incluir el punto.

Ejemplo de nombres de variables numéricas válidas son:

X

CANTIDAD

TOTAL

A34

B.1

Cada una de las variables numéricas anteriores contendrá un número, pero el ordenador sólo tomará su valor cuando hagamos referencia al nombre de la variable. Así, si en un momento dado, la variable X tiene el valor 5, cuando en el programa hagamos referencia a X, el ordenador operará directamente con su valor, 5, y podremos sumar el contenido de la variable X al de otro número o variable numérica.

A continuación vamos a ver nombres de variables numéricas inválidas y el porqué de su error:

3AH ---------- EMPIEZA POR UN DIGITO

C 1 ---------- CONTIENE UN BLANCO

E1HHV$A ---------- CONTIENE UN CARACTER QUE NO ES LETRA NI NUMERO

* VARIABLES ALFANUMERICAS

Son aquellas variables que pueden contener cualquier tipo de caracteres (letras, números u otros símbolos).

Para distinguir estas variables de las anteriores se termina el nombre de la variable con el carácter $. De esta manera tendremos:

CASA ------- VARIABLE NUMERICA (Contendrá un número)

CASA$ ------- VARIABLE ALFANUMERICA (Contendrá una serie de caracteres).

Mientras que con el contenido de la variable casa -numérica- se puede trabajar de forma aritmética (sumar, restar, etc.), con el contenido de la variable casa$ -alfanumérica-, al igual que ocurría con las constantes alfanuméricas, no es posible realizar operaciones aritméticas aunque lo que contengan sean números.

Ejemplos de nombres de variables alfanuméricas son: A$, D43$, F511$, nombre$, direccion$, ciudad$, etc.

2.3. Concepto de instrucción.

Podemos definir un programa como un *conjunto de órdenes codificadas en un lenguaje comprensible por el ordenador que, debidamente codificadas, resuelven un problema concreto.*

Cada una de las órdenes que componen un programa se llama *instrucción.*

A lo largo del texto se irán estudiando las instrucciones que componen el lenguaje BASIC. Hay instrucciones de introducción de datos, de salida de resultados, de operaciones con los datos, etc.

3. EL ORDENADOR EN MODO DIRECTO: COMANDO PRINT

El ordenador cuando trabaja en modo directo obedece de una manera inmediata la orden que se le da (en el momento que se pulsa <ENTER>). Estas órdenes que se realizan de forma inmediata se llaman *Comandos*.

3.1. Orden Print.

La orden PRINT sirve para que salga escrita en pantalla una constante, el valor de una variable o el resultado de la evaluación de una expresión numérica.

(Una expresión numérica es un conjunto de constantes y/o variables separadas por operadores aritméticos).

El comando PRINT sirve para utilizar el ordenador como una calculadora al evaluarnos expresiones.

Ejemplos:

Print 5+3 ----------> En pantalla aparecerá un 8

Print 8-1 ----------> En pantalla aparecerá un 7

Print 3*4 ----------> En pantalla aparecera un 12

Print 6/2 ----------> En pantalla aparecerá un 3

Aparte de los signos vistos en los ejemplos (iguales a los que se utilizan en matemáticas para las 4 operaciones básicas) existen dos operadores especiales en el teclado del Amstrad:

Indica división entera. (se puede utilizar con números menores que 32768).

Print 9/2 ----------> En pantalla aparecerá 4.5

Print 9\2 ----------> En pantalla aprecerá un 4

Indica exponenciación.

Print 2↑3 ----------> En pantalla aparecerá un 8

Print 4↑2 ----------> En pantalla aparecerá 16

LOS PARENTESIS

Cuando utilizamos el ordenador como calculadora nos es necesario muchas veces realizar varias operaciones juntas. ¿En qué orden se realizarán esas operaciones? Existe un orden de jerarquía entre los operadores de forma que primeramente se realizan las operaciones afectadas por los operadores de jerarquía más alta. Veamos cual es el orden de esas jerarquías:

1ª ↑

2ª * /

3ª + -

Entre operadores de igual jerarquía se empiezan realizando las operaciones por la izquierda.

Ejemplos

Print 5+4*3

Nos encontramos con dos operadores, + y *. Como la multiplicación tiene prioridad sobre la suma, primeramente se realiza la multiplicación

4*3=12

El resultado de la multiplicación, 12, se suma a 5

5+12=17

Por tanto 17 sería el resultado que nos aparecería en pantalla.

Print 8/2-1

En este caso se nos presenta una expresión con dos operadores, el cociente y la resta. Al tener el cociente una prioridad más alta que la resta se efectúa primeramente la división:

8/2=4

El resultado de la división, 4, será el minuendo que utilicemos en la resta.

4-1=3

El valor 3 sería el resultado que nos aparecería finalmente en pantalla.

Cuando se utilicen expresiones numéricas habrá que poner especial atención a las jerarquías de los operadores ordenando los números y los signos de forma adecuada.

Sin embargo hay veces que resultaría más cómodo cambiar el orden prefijado de los operadores. Esto lo podemos hacer mediante los *paréntesis*. Las operaciones encerradas entre paréntesis son las que tienen mayor prioridad. (Si existiesen varias operaciones dentro de un mismo paréntesis el orden de realización de éstas vendría dado por la jerarquía de operadores vista anteriormente).

Ejemplos.

Siguiendo estos ejemplos, podemos observar como evalúa el ordenador expresiones numéricas.

```
Print 5 * (3 + 4)
```

Primeramente se realiza la operación entre paréntesis

```
3+4=7
```

El valor hallado se multiplica por 5.

```
5*7=35
```

El número 35 es el resultado final. Podemos comprobar que la misma expresión sin paréntesis daría como resultado 19.

```
Print 4 * (2 + 3) - 5 / (2 + 3 * 1)
```

En este ejemplo vemos que existen dos paréntesis. Empecemos resolviendo el primero

```
2+3=5
```

La expresión resultante es `4 * 5 - 5 / (2 + 3 * 1)`

Continuemos con el segundo paréntesis:

```
2+3*1
```

Dentro de este paréntesis nos encontramos con un producto y una suma. Empezamos realizando el producto.

```
3*1=3
```

Nos quedaría la expresión

```
4 * 5 - 5 / (2 + 3)
```

Seguimos resolviendo el paréntesis

2+3=5

La nueva expresión resultante sería

4 * 5 - 5 / 5

En esta expresión hay un producto, un cociente y una diferencia. Como el producto y el cociente tienen la misma jerarquía, comenzamos por el que está mas a la izquierda, el producto.

4*5=20

La expresión se simplifica quedándonos

20 - 5 / 5

La división tiene más preferencia que la resta por tanto realizaremos el cociente

5/5=1

Después nos faltaría por operar

20 - 1

Obteniendo finalmente el resultado 19.

Opciones de Presentación.

Hemos visto que mediante la orden PRINT se puede escribir una constante o el valor de una variable. También podemos escribir varias constantes o el valor de varias variables con una única instrucción PRINT. Para ello debemos separar las constantes o variables por ";" o por ",".

Si se separan las constantes y/o variables por ";" aparecerán juntas en pantalla mientras que si la separación es mediante "," habrá 12 espacios en blanco entre cada una de ellas.

Ejemplo.

```
Print "Mi Santo es el día"; 1
```

En pantalla aparecerá:

Mi Santo es el día 1

(El ordenador deja un espacio antes de los números para el signo. Si es un número positivo deja un espacio en blanco).

```
Print "La capital de Francia es", "París"
```

Apareciendo en pantalla:

La capital de Francia es París

(----------): 12 espacios en blanco

Otro recurso para dejar espacios en blanco es la función TAB seguida de un número entre paréntesis. Esta función nos empieza a escribir en el número de la columna que se indique.

Ejemplo

```
Print tab (10);"hola"
```

En pantalla debe aparecer

```
      hola
        |
        |
        ↓
   Columna 10
```

La orden LPRINT actúa igual que PRINT pero escribiendo en papel en vez de en pantalla.

3.2. El Código ASCII. ASC Y CHR$

Cada carácter definido en el ordenador tiene un número que lo identifica. Este número se llama código ASCII de ese carácter. Es decir el código ASCII hace corresponder un número con cada carácter.

La instrucción ASC nos proporciona el código ASCII correspondiente a un determinado carácter. Supongamos que queremos saber el código ASCII de la letra *a*. Escribiremos:

```
PRINT ASC ("a")
```

En pantalla nos aparecerá el número 97 que indica que el código ASCII correspondiente al carácter *a* es el número 97.

De un modo similar se puede obtener un carácter a partir de su código ASCII mediante la instrucción CHR$. Si ahora escribimos:

```
PRINT CHR$(97)
```

El ordenador escribe la letra *a*.

Podemos ver todos los caracteres definidos en el ordenador teniendo en cuenta que los caracteres imprimibles (hay otros caracteres que son de control) tienen como código ASCII los números comprendidos entre 32 y 255. Tendremos que escribir las siguientes instrucciones:

```
10 FOR i=32 to 255
20   PRINT CHR$(i);i
30 NEXT i
```

Al lado de cada carácter aparece escrito su código ASCII correspondiente, de modo que si queremos escribir una "u" acentuada (código 228) podemos escribir:

```
PRINT CHR$(228)
```

Veremos que aparece la letra "ó" en la pantalla.

3.3. Limpiar la pantalla.

Los caracteres cuyos códigos ASCII están comprendidos entre 32 y 255 son caracteres imprimibles en pantalla como hemos visto. Los códigos de número menor que 32 corresponden a caracteres de control del ordenador (pantalla, sonidos, etc.). Una de las operaciones más habituales en un programa es borrar la pantalla. Esta acción se consigue con la siguiente secuencia:

```
PRINT CHR$(27)+"E";CHR$(27)+"H"
```

También se puede definir en una variable alfanumérica la secuencia de borrado y posteriormente escribir la variable.

Ejemplo:

```
a$=CHR$(27)+"E"+CHR$(27)+"H"

PRINT a$
```

La ventaja de este segundo método es poder dar órdenes de impresión de la variable cada vez que se quiera borrar la pantalla, sin necesidad de escribir repetidamente los códigos.

4. SENTENCIAS Y COMANDOS

4.1. Diferencia entre comando e instrucción.

En el capítulo anterior se veía cómo tecleando el comando PRINT con algún argumento se ejecuta la orden asociada a ese comando (imprimir algo en pantalla).

Podemos unir una serie de órdenes y que se ejecuten todas juntas cuando nos interese. Habremos formado un programa. Las órdenes que componen un programa ya no se llaman comandos sino *sentencias* o *instrucciones*. El orden en que se ejecutan éstas viene dado por el número que precede a cada sentencia. De modo que instrucciones que tengan un número pequeño se ejecutarán antes que aquellas que vayan precedidas de un número más alto. Si bien existen excepciones a esta regla que ya estudiaremos.

Vemos entonces la diferencia entre COMANDO e INSTRUCCION: el comando no lleva número de línea y el ordenador lo ejecuta inmediatamente (en cuanto se teclea <ENTER>), en cambio las instrucciones se agrupan y forman un programa.

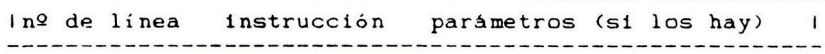
Cualquier comando se convierte en instrucción cuando le anteponemos un número de línea. Entonces ya no se ejecutará directamente cuando tecleemos <ENTER> sino que se almacenará en memoria y sólo tendrá efecto cuando tecleemos el programa entero.

4.2. Formato de la Sentencia Basic.

En BASIC cada instrucción lleva un número de línea. Este número ha de ser entero. Si en un mismo número de línea se quieren agrupar varias instrucciones, éstas deben ir separadas por dos puntos (:).

A continuación del número de línea se escribe la instrucción que es una palabra reservada del lenguaje BASIC.

Por último y dependiendo del tipo de instrucción se escriben los parámetros de la instrucción. Los parámetros pueden ser uno, varios o ninguno.



Formato de una sentencia en BASIC.

Ejemplo

```
10    PRINT 14
```

Si se teclea esta instrucción se comprueba que, en principio, no se escribe el número 14. Aparentemente no pasa nada. Sin embargo el ordenador ha introducido en memoria ésta instrucción.

4.3. Introducción de un programa.

Para introducir un programa en el ordenador se deben ir tecleando cada una de las instrucciones. No hay que olvidar pulsar la tecla <ENTER> entre dos instrucciones consecutivas.

Los números de instrucción irán de menor a mayor (para que se ejecuten en ese orden) y se deben dejar varios números de diferencia entre una instrucción y la siguiente pues si queremos posteriormente introducir una instrucción entre dos que hayamos tecleado no tendremos más que anteponerla un número intermedio.

4.4. Ejecución de un programa: COMANDO RUN

Vamos a introducir un programa y después lo ejecutaremos. El programa va a consistir en realizar un rectángulo a base de asteriscos.

El programa, cuando se ejecute, empezará borrando lo que hubiera anteriormente en la pantalla.

```
10 REM RECTANGULO

20 PRINT "**********"

30 PRINT "*        *"

40 PRINT "*        *"

50 PRINT "**********"

60 PRINT "Rectangulo"
```

Ya tenemos introducido el programa en memoria. Ahora vamos a ejecutarlo, es decir a pedir al ordenador que realice cada una de las instrucciones que componen el programa. Para ello escribimos

```
RUN
```

Después de escribir este comando y de pulsar <ENTER>, en pantalla aparecerá:

```
**********
*        *
*        *
**********
Rectangulo
```

4.5. Listar un programa: Comando LIST.

Después de ejecutar el programa quizá queramos añadir o modificar alguna instrucción, o simplemente ver las instrucciones que hemos introducido. Disponemos de un comando que *lista las instrucciones que componen un programa*. Es el comando LIST.

Teclea

```
LIST
```

y cuando pulses <ENTER> aparecerá

```
10 REM RECTANGULO
20 PRINT "**********"
30 PRINT "*        *"
40 PRINT "*        *"
50 PRINT "**********"
60 PRINT "Rectangulo"
```

4.6. Corrección de Líneas: Comando EDIT.

Imaginemos que queremos realizar dos rectángulos pequeños a partir del que tenemos. Tendremos que modificar las instrucciones 20 y 30. Para hacerlo utilizamos el comando EDIT seguido del número de Línea que queremos cambiar:

```
EDIT 30
```

En pantalla aparecerá lo que teníamos en la instrucción 30

```
EDIT 30
PRINT "*          *"
```

Con los cursores ← y → posicionamos el cursor en el blanco siguiente al primer asterisco e introducimos asteriscos en lugar de los blancos que los borramos quedándonos después de la operación

```
EDIT 30
PRINT "**********"
```

Igualmente haríamos con la instrucción 40, escribiendo EDIT 40 y procediendo del mismo modo con los cursores.

4.7. Añadir Líneas.

Para añadir instrucciones al programa se introducen con su número correspondiente y se ordenan automáticamente según este número.

Hay que prestar especial atención para *no insertar instrucciones con números de línea ya existentes* pues se perderían las instrucciones que tuvieran anteriormente esos números de línea.

Imaginemos que en el ejemplo del rectángulo queremos que al principio aparezca en pantalla el mensaje:

Dibujo de un rectángulo

Tendríamos que situar entre la instrucción 10 y la instrucción 20 la siguiente instrucción:

PRINT "Dibujo de un rectángulo"

Esta instrucción deberá ir precedida de un número entre 10 y 20, por ejemplo el 15. Escribiríamos por tanto

15 PRINT "Dibujo de un rectángulo"

Automáticamente se situará esta instrucción en el lugar deseado como podemos comprobar si listamos el programa (comando LIST). Veríamos:

```
10 REM RECTANGULO
15 PRINT "Dibujo de un rectángulo"
20 PRINT "**********"
30 PRINT "*        *"
40 PRINT "*        *"
50 PRINT "**********"
60 PRINT "Rectangulo"
```

4.8. Borrar una línea.

Hemos visto en el apartedo anterior como se pueden sustituir instrucciones insertando sentencias con números de instrucción ya existentes. *Si escribimos un número de línea de una instrucción ya existente y pulsamos <ENTER>, sin introducir ninguna otra instrucción, habremos borrado la instrucción que tiene ese determinado número de línea.*

Podemos borrar la línea 15 que hemos insertado en el apartado anterior escribiendo

15 <ENTER>

Listando el programa quedará

```
10 REM RECTANGULO
20 PRINT "**********"
30 PRINT "*        *"
40 PRINT "*        *"
50 PRINT "**********"
60 PRINT "Rectangulo"
```

4.9. Borrar Varias Líneas: COMANDO DELETE.

Para borrar varias líneas podemos seguir el procedimiento del apartado anterior, borrando instrucciones una a una. Pero si las instrucciones que se quieren borrar tienen números de línea consecutivos existe un procedimiento más rápido que es utilizando el comando DELETE seguido del número de la primera instrucción que se quiere borrar y por el numero de la última instrucción que se quiere suprimir del programa. Estos dos números se separan por un guión.

Siguiendo con el ejemplo del rectángulo, borremos las tres primeras instrucciones. Tendremos que escribir

```
DELETE 10-30
```

Listando ahora el programa veremos:

```
40 PRINT "*        *"
50 PRINT "**********"
60 PRINT "Rectangulo"
```

4.10. Borrar un Programa en Memoria: Comando NEW.

Si un programa que tenemos en memoria ya no es útil, será necesario borrarlo de la misma. La solución drástica es desconectar el sistema de alimentación del ordenador. Sin embargo existe un método más elegante que es utilizando el comando NEW. Si escribimos:

NEW

habremos borrado el programa como podemos comprobar si intentamos listarlo.

Existe otro procedimiento que inicializa o resetea totalmente el ordenador, consiste en pulsar simultáneamente:

<MAYS> <EXTRA> <SAL>

Este método tiene los mismos efectos que desenchufar el ordenador. El comando NEW se diferencia en que borra un programa pero no la pantalla.

4.11. Renumeración de Líneas.

Al escribir un programa, es fácil que olvidemos algunas instrucciones y que tengamos que intercalarlas en el programa al final o bien hacerlo sobre la marcha. Si las hemos numerado de 10 en 10 y resulta que entre las líneas 40 y 50 hemos intercalado la 45 y posteriormente entre la 45 y la 50 intercalamos varias (46, 47, 48...) tendremos un programa poco legible.

10 ----------

20 ----------

30 ----------

```
40 ----------------
45 ----------------
46 ----------------
47 ----------------
48 ----------------
50 ----------------
"
etc.
```

Con el comando RENUM podemos renumerar las líneas de un programa de 10 en 10 y empezando por la instrucción 10. Existen varias opciones según el formato que le demos pues también podemos renumerarlo a partir de un número de línea determinado.

El formato de la instrucción es:

RENUM X, Y, Z

-X será el nuevo número de línea a partir del cual se va a renumerar el programa.

-Y representa el anterior número de línea, es decir, aquel que va a ser sustituído por un nuevo número (X).

-Z indica el salto entre líneas.

Es decir, renumera las líneas de un programa a partir del número de línea antiguo (Y), al cual se hace corresponder el número de línea nuevo (X). El tercer número o parámetro (Z), indicará el salto o incremento entre líneas.

Si se omite el tercer número o parámetro, las líneas quedan renumeradas de 10 en 10.

Si omitimos el número de línea viejo (Y), se renumera a partir del primer número de línea del programa.

Por último, si no damos un número como primer parámetro (X), la renumeración comienza por 10.

Ejemplo

```
RENUM
```

En este caso, al no darle ningún parámetro, este comando escrito al final del programa nos dará como resultado uno nuevo, cuya numeración empezará por 10 y las líneas se irán incrementando de 10 en 10. De este modo, en nuestro ejemplo anterior, una vez utilizado el comando **RENUM** y listado el programa irán todas las instrucciones de 10 en 10.

ANTES	AHORA
10 ----------	10 ----------
20 ----------	20 ----------
30 ----------	30 ----------
40 ----------	40 ----------
45 ----------	50 ----------
46 ----------	60 ----------
47 ----------	70 ----------
48 ----------	80 ----------
50 ----------	90 ----------
"	"
"	"

El ejemplo anterior también lo podíamos haber renumerado de la siguiente manera:

RENUM 50,45,10

El resultado de la renumeración sería el mismo que el visto anteriormente.

De igual forma quedaría si omitiésemos el tercer parámetro (10)

RENUM 50,45

También podíamos omitir el número de línea antiguo:

RENUM 20,,20

En este caso, se comenzaría a renumerar desde el primer número de línea antiguo, sustituyendo este por el nuevo número expresado:

ANTES	AHORA
10 ----------	20 ----------
20 ----------	40 ----------
30 ----------	60 ----------
40 ----------	80 ----------
45 ----------	100 ----------
46 ----------	120 ----------
47 ----------	140 ----------
48 ----------	160 ----------
50 ----------	180 ----------
"	"

5. ASIGNACION DE VARIABLES.

Las instrucciones de asignación son habitualmente las que más se repiten en un programa. Recordemos que las variables son unidades de memoria que tienen un nombre y contienen un valor. Con estas instrucciones podremos dar un valor a una variable.

5.1. Instrucción LET.

La instrucción LET asigna un valor a una variable. En algunos ordenadores es una sentencia opcional y es indiferente utilizarla.

LET <variable> = constante o variable o expresión numérica.

Ejemplos

X=4 ---------- Con esta instrucción damos a la variable X el valor 4.

LET X=4 ---------- Idem que la anterior.

a$="direccion" ------ Se asigna a la variable a$ la constante alfanumérica dirección.

LET F=H ---------- La variable F toma el contenido de la variable H.

C=3*4+5 ---------- Se evalúa la expresión 3*4+5 y su resultado, 17, se asigna a la variable C.

5.2. Funciones Matemáticas.

Muchas aplicaciones que se realizan en el ordenador son para resolución de problemas matemáticos. El lenguaje BASIC

nos ofrece algunas funciones incorporadas que potencian la capacidad científica de este lenguaje.

Función SQR.

La función SQR realiza la raíz cuadrada de una constante o variable numérica.

Ejemplo

```
PRINT SQR(9)
```

En pantalla aparecerá el número 3.

Función ROUND.

Esta Función redondea un número decimal al entero más cercano.

```
PRINT ROUND (1.8)-------> Nos escribe un 2.
PRINT ROUND (1.3)-------> Nos escribe un 1.
```

Función ABS.

La Función ABS proporciona el valor absoluto de un número entero.

Ejemplo.

```
PRINT ABS (-7)-------> Nos escribe un 7
```

Función STR$

La función STR$ convierte a alfanumérico el contenido numérico de una variable o expresión numérica.

Ejemplo

```
a=7
a$=STR$(a)
```

Función VAL

La función VAL es la inversa de la función STR$. Convierte en numérico el contenido alfanumérico de una expresión o variable alfanumérica. Si aquel carece de valor numérico genera el valor 0.

```
a$="4038954"
a=val (a$)
```

Con el contenido de la variable *a* -4038954- se pueden realizar operaciones aritméticas.

5.3. Input.

Supongamos que queremos hacer un programa que calcule el área de un cuadrado de lado 5. Escribiremos:

```
10 LET lado=5

20 LET area=lado * lado

30 PRINT area
```

El programa, en principio, satisface nuestras necesidades. Sin embargo, si quisiéramos ahora calcular el área de un cuadrado, cuyos lados tuvieran otra medida, no nos serviría ese programa. Tendríamos que modificar la instrucción 10 y dar a la variable lado el nuevo valor.

Este problema lo podemos solucionar con *la instrucción Input que permite asignar un valor a una variable en tiempo de ejecución*. Esto da una gran versatilidad a los programas.

El ordenador, cuando ejecuta una instrucción INPUT, nos da opción a rellenar desde el teclado la variable que

nosotros hayamos puesto a continuación de la instrucción INPUT. Además, esta asignación externa se detecta porque en la pantalla aparece un signo de interrogación indicándonos con ello que podemos teclear a continuación el valor de la variable.

Si en el programa anterior, que calcula el área de un cuadrado, utilizamos la instrucción INPUT podremos calcular el área de cualquier cuadrado. Escribiríamos:

```
10 INPUT lado
20 LET area=lado*lado
30 PRINT area
```

Al ejecutarse este programa aparece en pantalla una interrogación (?) y se detiene la ejecución hasta que le introduzcamos un número. Este número va a ser el contenido de la variable que sigue a la instrucción INPUT. En nuestro ejemplo, si quisiéramos calcular el área de un cuadrado cuyo lado mide 7, introduciríamos un 7. En cada ejecución podemos introducir un número distinto, pudiendo calcular tantas áreas de cuadrados como veces ejecutemos el programa. De ahí la versatilidad de la que antes hablábamos.

5.4. Funciones Definidas por el Usuario

En el apartado 5.2. se estudió un conjunto de funciones que matemáticas que proporciona el BASIC. El usuario también puede definir sus propias funciones pudiéndolas adecuar a las necesidades de su programa.

Para definir una función se escribe la instrucción:

```
DEF FN nombre(arg1, arg2,...argn)=f(arg1, arg2,...argn)
```

En esta expresión cabe distinguir:

* DEF FN: Es la orden de definición de una función.

* nombre: Es el identificativo de la función. En el caso de que la función devuelva un valor alfanumérico deberá terminar en el carácter "$"

* arg1,arg2...argn: Es la lista de argumentos. El resultado que devuelve la función está en función de ellos.

* f(arg1, arg2, ... argn): Indica como están enlazados los argumentos que intervienen en la función.

Ejemplo

Realizar una función que calcule el área de un rectángulo.

En este caso el nombre de la función deberá ser el de una variable numérica, ya que el resultado que esperamos obtener de la función es numérico (el área). Los argumentos van a ser la base y la altura del rectángulo. Por último, la forma de enlazar esos argumentos es multiplicándolos.

Escribiremos por tanto:

```
DEF FN area(base,altura)=base*altura
```

Una vez que tenemos definida una función en nuestro programa podemos utilizarla como si fuera una función predefinida (como SQR o ROUND por ejemplo).

Si queremos calcular el área de un rectángulo que tenga de base 6 cm. y de altura 3 cm. escribiremos:

```
PRINT FN area(6,3)
```

El valor 6 irá a parar a la variable base y el valor 3 a la variable altura, con lo que el programa generará el valor 18.

6. MEJORANDO LA PRESENTACION.

En este capítulo estudiaremos una serie de instrucciones que permiten mejorar las presentaciones de las pantallas, cuando se ejecutan los programas, así como la legibilidad de los listados cuando queremos ver las instrucciones que componen un programa.

6.1. Pantallas más claras: Ubicación del Cursor

Existe una secuencia de caracteres de control para situar el cursor en un punto dado de la pantalla y poder escribir un texto en la posición deseada. La forma más sencilla es definir una función con dos argumentos, la fila y la columna donde queramos que aparezca el cursor.

Escribiremos:

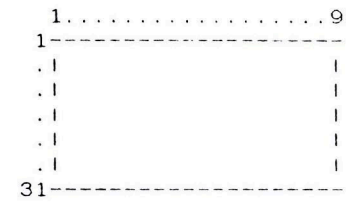
```
DEF FN cursor$(y,x)=CHR$(27)+"Y"+chr$(32+y)+chr$(32+x)
```

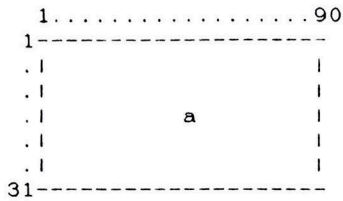
A partir de esta instrucción cuando queramos situar el cursor en un determinado punto de la pantalla escribiremos:

```
PRINT FN cursor$(altura, anchura)
```

Siendo altura y anchura los números que interese utilizar como coordenadas.

La configuración de la pantalla es la siguiente:





Si queremos que aparezca el carácter "a" en el centro de la pantalla (altura=15, anchura=45), escribiremos:

```
10 DEF FN cursor$(y,x)=CHR$(27)+"Y"+chr$(32+y)+chr$(32+x)

20 PRINT FN cursor$(15,45);"a"
```

6.2. Listados más claros: Instrucción REM.

Los programas deben documentarse. *Documentar un programa es escribir una serie de características asociadas al programa*, tales como:

Qué hace

Quién es el autor

Fecha de Realización

Aplicaciones del Programa

...

Toda esta información deberá aparecer al listar un programa. También, cuando un programa es largo, es conveniente que aparezcan explicaciones contando para lo que sirve cada parte del programa. De esta manera si con el paso del tiempo hay que realizar alguna modificación será más sencillo hacerlo, especialmente si la persona que modifica el programa no es su autor.

Este grupo de instrucciones cuyo único cometido es aportar información comienzan por la palabra

REM

(REM viene de Remarks, en inglés, comentario)

Ejemplo.

```
10 REM Este programa es un ejemplo de la instrucción Rem
20 REM El programa borra la pantalla y escribe en la par-
       te superior de la pantalla un texto.
30 PRINT CHR$(27)+"E";CHR$(27)+"H"
40 REM Con la instrucción 30 se borra la pantalla.
50 PRINT "Estoy aprendiendo la instruccion REM"
60 REM Aquí acaba el programa.
```

El programa anterior, a efectos prácticos, podía haber quedado reducido a dos instrucciones (las que no son REM) pero habríamos perdido información al leer el listado del programa.

Programa equivalente sin REM

```
30 PRINT CHR$(27)+"E";CHR$(27)+"H"
50 PRINT "Estoy aprendiendo la instruccion REM"
```

6.3. Fin del programa: Instrucción END.

Los programas que hemos visto hasta ahora acaban al ejecutarse la última instrucción. Para mayor claridad podemos poner después de la última instrucción la instrucción END. Como veremos habrá ocasiones en que es imprescindible el uso de la instrucción END. Será cuando esta instrucción no figure al final de un programa, sino como una instrucción intermedia.

7. CONTROLANDO EL PROGRAMA.

7.1. Bifurcación Incondicional: GOTO

Según se ha visto anteriormente, el orden de las líneas de instrucción dentro de un programa viene dado por el número de línea que las acompaña. En consecuencia, una línea precederá a otra si su número de línea es inferior.

En algunos casos, es conveniente que el programa no se ejecute de una forma totalmente secuencial, sino que podemos ejecutar tras una cierta instrucción otra que esté en otro punto del programa. Esto lo hacemos mediante la instrucción GOTO.

La ruptura de secuencia de ejecución de un programa puede ser desde luego obligatoria (incondicional); no obstante, también es posible que la ruptura no deba realizarse en cualquier caso, sino que dependa del cumplimiento de alguna condición impuesta (más adelante veremos esta posibilidad).

La sintaxis de una instrucción GOTO es

```
GOTO n
```

donde n representa el número de línea de la instrucción a la que se quiere saltar.

Ejemplo.

Vamos a realizar un programa para escribir los números impares empezando por 1.

```
10 LET n=1
20 PRINT n
30 LET n=n+2
40 GOTO 20
```

En el programa anterior, n comienza valiendo 1, y se escribe su valor, a continuación se incrementa su valor en 2 y entonces vale 3. Con la instrucción 40, el programa salta a la instrucción 20 donde se manda que se escriba el nuevo valor de n, 3. Este ciclo se va repitiendo y n va tomando los valores impares que se van escribiendo.

Tal como hemos realizado este programa, su ejecución no tiene fin, pues cuando llega a la última instrucción vuelve a saltar a la instrucción 20. Para hacerlo acabar nos veremos obligados a pulsar dos veces la tecla <ESC> que aborta el programa (interrumpe su ejecución) escribiendo un mensaje del tipo:

Break in n

Donde n es el número de línea de la instrucción que se estaba ejecutando en el momento de pulsar la tecla <ESC>.

Podemos conseguir que el programa continúe ejecutándose escribiendo la palabra

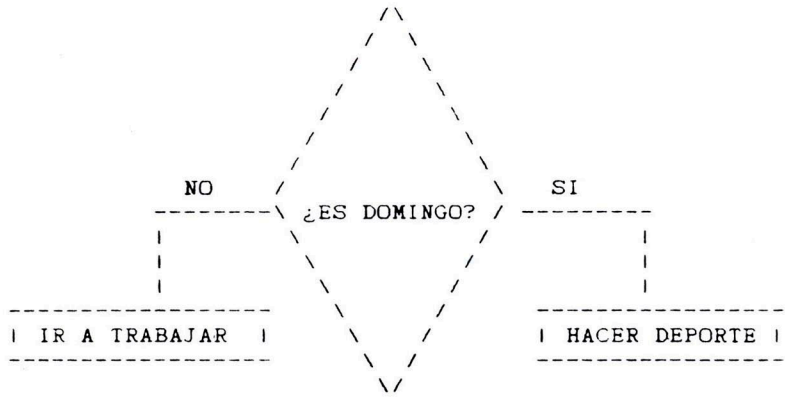
CONT

y el programa continuará ejecutándose desde el punto donde se abortó.

7.2. Condiciones.

Las condiciones son muy usadas en un programa. En función de que se cumpla o no una condición se ejecutarán una serie de instrucciones u otras.

Las condiciones actúan en un programa de modo similar a la conducta humana. Pensemos en una persona que trabaja todos los días menos los domingos, día en que va a hacer deporte. Podemos hacer un organigrama (un organigrama es una representación gráfica de la resolución de un problema) estudiando su conducta.



7.3. La Bifurcación Condicional: IF/THEN

En BASIC, como en la mayor parte de los lenguajes, las condiciones se expresan mediante la instrucción IF. La instrucción IF va asociada a otra instrucción, la instrucción THEN, que indica que acción hay que tomar si se sumple la condición. El formato de la instrucción es:

IF condicion THEN ejecución de alguna instrucción

De no cumplirse la condición, la ejecución del programa salta inmediatamente a la instrucción cuyo número es el siguiemnte a la instrucción IF.

Ejemplo

Supongamos que en el programa que halla los números impares, en el apartado 7.1. queremos que sólo nos escriba los números impares hasta el número 100. Escribiríamos:

```
10 LET n=1
20 PRINT n
30 LET n=n+2
40 IF n<100 THEN GOTO 20
```

Este programa sí tiene fin, pues cuando n sea mayor que 100 ya no se cumplirá la condición expresada en la instrucción 40. Entonces el programa saltará a la instrucción siguiente a la 40 y como ya no existen más instrucciones termina la ejecución.

7.4. Más Sobre Condiciones: ELSE

Hay veces que es necesario ejecutar un conjunto de instrucciones si se cumple una condición. En este caso utilizamos la bifurcación condicional antes vista. Pero en ocasiones, si no se cumple la condición tenemos que ejecutar otras instrucciones. Es decir que se cumpla o no la condición se impondrán unas decisiones (si bien éstas serán distintas).

Para ello contamos con la opción ELSE dentro de una instrucción IF, de tal modo que si al evaluar la condición ésta se cumple, se ejecutará la/s sentencia/s que vaya/n detrás de la palabra THEN (recordemos que pueden ir varias sentencias separadas de dos puntos). De no cumplirse, se ejecuta el conjunto de instrucciones que sigan a la palabra reservada ELSE.

Ejemplo

Programa para simular una máquina expendedora de billetes de Metro.

```
10 CLS$=CHR$(27)+"E"+CHR$(27)+"H"
20 INPUT "Precio del Billete";pb
30 INPUT "Número de billetes";nb
```

```
40 PRINT CLS$

50 INPUT "Monedas de 50";m50

60 INPUT "Monedas de 25";m25

70 INPUT "Monedas de 5";m5

80 LET dinero=50*m50 + 25*m25 +5*m5

90 LET pagar=nb*pb

100 IF dinero<pagar THEN PRINT "Dinero Insuficiente" ELSE
    LET vuelta=dinero-pagar:PRINT CLS$:PRINT "A
    devolver";vuelta

110 END
```

En este programa si el dinero introducido es menor que el dinero que hay que pagar en pantalla aparecerá *Dinero Insuficiente*. Si no es así se ejecutan las instrucciones que van detrás de la palabra ELSE.

7.5. Condiciones compuestas: AND y OR

Hasta ahora hemos visto que la instrucción IF provoca que se ejecuten unas u otras instrucciones según se cumpla o no una condición. Hay veces que es necesario que se cumpla una combinación de condiciones para que se ejecute una instrucción. Estas combinaciones se enlazan mediante los operadores lógicos AND y OR.

AND

El operador AND sirve para enlazar varias condiciones de forma que se evaluará el resultado como cierto si se cumplen todas las condiciones.

Ejemplo

Programa para saber si un número es divisible por 6.

Un número es divisible por 6 si lo es por 2 y por 3.

Por tanto, en este ejemplo se tienen que cumplir 2 condiciones:

1. Que el número sea divisible por 3.
2. Que el número sea divisible por 2.

```
10 INPUT "Introducir número";n
20 IF (INT (n/2)=n/2 AND INT (n/3)=n/3) THEN PRINT
   "Es un número divisible por 6" ELSE PRINT "No es
    un número divisible por 6"
```

OR

El operador OR sirve para enlazar varias condiciones de modo que se evaluará el resultado como cierto si se verifica alguna de las condiciones.

Ejemplo

Se desea saber si una persona tiene que ir o no a clase un determinado día.

Hay tres condiciones para que una persona tenga vacación:

1. Que sea sábado.
2. Que sea domingo.
3. Que sea festivo.

```
10 INPUT "Es sábado";a$
20 INPUT "Es domingo";b$
30 INPUT "Es festivo";c$
40 IF a$="SI" OR b$="SI" OR c$="SI" THEN PRINT "Vd. hoy
```

```
no tiene que ir a clase" ELSE PRINT "Vd. debe acudir a
clase"

50 END
```

¿Qué pasaría si la persona que ejecuta el programa tecleara la palabra SI con minúsculas?¿Qué se puede hacer?

7.6. Bifurcación Múltiple Condicional. ON/GOTO

A veces una condición puede tener múltiples alternativas. Existe una instrucción que en función del valor de una variable bifurca a distintas partes del programa. Es la instrucción ON.

Sintaxis:

ON expresión entera GOTO n1,n2,n3...

Donde n1, n2, n3 son números de línea de forma que si la expresión entera vale 1 la ejecución del programa continuará por la instrucción n1; si vale 2, el programa seguirá por n2; si vale 3, el programa seguirá por n3 y así sucesivamente. En caso de que el resultado de la expresión entera no esté comprendido entre 1 y el número de valores, el programa salta a la instrucción siguiente.

Ejemplo

Programa que nos dice el día de la semana

```
10 INPUT "Introducir número de día";d

20 On d goto 40,50,60,70,80,90,100

30 PRINT "Día Inválido":GOTO 110

40 PRINT "Lunes":GOTO 110

50 PRINT "Martes":GOTO 110
```

```
60 PRINT "Miércoles":GOTO 110
70 PRINT "Jueves":GOTO 110
80 PRINT "Viernes":GOTO 110
90 PRINT "Sábado":GOTO 110
100 PRINT "Domingo"
110 END
```

8. MANEJANDO LETRAS.

Con los caracteres que forman una palabra se puede realizar una serie de operaciones mediante ciertas instrucciones que estudiaremos en este capítulo.

8.1. Concatenar Caracteres.

Concatenar caracteres es unir al final de una cadena de caracteres, otra cadena. De esta manera de dos palabras podemos obtener otra palabra compuesta de las dos anteriores. Se unen con el signo "+".

Ejemplo:

Unir la palabra balonmano a partir de "balon" y de "mano".

```
10 LET a$="balon"
20 LET b$="mano"
30 LET c$=a$ + b$
40 PRINT c$
```

8.2. Longitud de una Cadena: Instrucción LEN

La instrucción LEN cuenta el número de caracteres que componen una cadena alfanumérica.

Ejemplo

Calcular el número de caracteres que componen la palabra casa.

```
10 LET a$="casa"
```

```
20 PRINT LEN (a$)
```

La longitud de la cadena a$ es de 4 caracteres.

8.3. Sacar Caracteres de la Izquierda: LEFT$.

La instrucción LEFT$ toma un número determinado de caracteres comenzando por el primero de la cadena.

Sintaxis:

```
LEFT$(a$,n)
```

a$: Variable que contiene la palabra de la que se quieren extraer los n primeros caracteres.

Ejemplo

Escribir las tres primeras letras de la palabra "Fernando"

```
10 LET a$="Fernando"
20 b$=LEFT$(a$,3)
30 PRINT b$
```

En pantalla aparecería *Fer*.

8.4. Sacar Caracteres de la Derecha: RIGTH$.

La instrucción RIGHT$ toma un número determinado de caracteres comenzando por el final de la cadena.

Sintaxis:

RIGTH$(a$,n)

a$: Nombre de la variable de la cual se quieren extraer los n últimos caracteres que contiene.

Ejemplo

Escribir los 2 últimos caracteres de la palabra "Pisuerga"

```
10 a$="Pisuerga"
20 b$=RIGHT$(a$,2)
30 PRINT b$
```

Después de ejecutar estas instrucciones, en pantalla aparece *ga*.

8.5. Extraer Caracteres Intermedios.

Para extraer los caracteres intermedios de una cadena se utiliza la instrucción MID$.

Sintaxis:

MID$(a$,n,m)

a$: Variable de la cual se quieren obtener los m primeros caracteres comenzando a contar desde el que ocupa la posición n.

Ejemplo:

Si queremos extraer dos caracteres a partir del 3º en la palabra Madrid escribiremos:

```
10 a$="Madrid"
20 b$=MID$(a$,3,2)
30 PRINT b$
```

En pantalla aparecerá *dr*.

8.6. Conversión a Mayúsculas: UPPER$.

Si queremos convertir el contenido de una variable alfanumérica a mayúsculas deberemos utilizar la instrucción UPPER$.

Sintaxis:

```
UPPER$(a$)
```

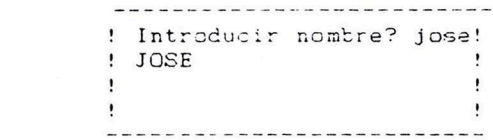
a$: Variable cuyo contenido se quuiere pasar a mayúsculas.

Ejemplo

Introducir el nombre de una persona y que en pantalla aparezca en mayúsculas.

```
10 INPUT "Introducir nombre";n$
20 a$=upper$(nombre$)
30 PRINT a$
```

En pantalla aparecería:



8.7. Conversión a Minúsculas. Instrucción LOWER$.

Esta instrucción es la inversa de la instrucción anterior. Convierte a minúsculas el contenido de una variable alfanumérica.

Sintaxis:

```
LOWER$(a$)
```

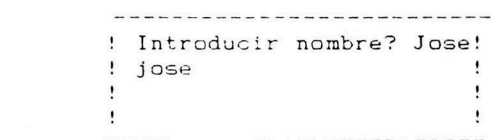
a$: Variable cuyo contenido se quiere pasar a mayúsculas.

Ejemplo

Introducir el nombre de una persona y que en pantalla aparezca en minúsculas.

```
10 INPUT "Introducir nombre";n$
20 a$=LOWER$(n$)
30 PRINT a$
```

En pantalla aparecería:



Como aplicación de las instrucciones vistas en este capítulo vamos a realizar un programa en el que se introduce un nombre y en pantalla aparecen las letras de ese nombre en minúsculas excepto la primera que deberá salir en mayúsculas.

```
10 REM Ejemplo de manejo de cadenas

20 INPUT "Introducir nombre";n$

30 LET numcar=LEN(n$)

40 REM En la variable numcar tenemos el número de
     caracteres

50 LET ini$=LEFT$(n$,1)

60 REM En la variable ini$ tenemos la letra inicial
   de la palabra que hemos introducido

70 LET inimay$=UPPER$(ini$)

80 REM La variable inimay$ contiene la letra ini-
   cial de la palabra en mayúscula.

90 LET resto$=RIGHT$(n$,numcar-1)

100 REM La variable resto$ contiene todas las le-
    tras introducidas menos la primera.

110 LET restomin$=LOWER$(resto$)

120 REM La variable  restomin$ contiene todas las
    letras introducidas, menos la primera, en
    minúsculas

130 LET result$=inimay$+restomin$

140 REM La variable result$ une la inicial en ma-
    yúsculas con el resto de letras en minúsculas.

150 PRINT result$

160 REM Se imprime resultado final.
```

9. REPITIENDO INSTRUCCIONES.

9.1. Concepto de Bucle.

Hay veces que es necesario repetir varias veces un conjunto de instrucciones. Supongamos que queremos escribir los 10 primeros números. Un procedimiento sería:

```
10 PRINT 1
20 PRINT 2
30 PRINT 3
...
100 PRINT 10
```

Como se puede observar el método es bastante tedioso. Lo podemos simplificar con una combinación de las instrucciones IF y GOTO ya estudiadas. El programa quedaría como sigue:

```
10 numero=1
20 PRINT numero
30 numero=numero+1
40 IF numero<11 then GOTO 20
```

En este ejemplo, cada vez que se escribe el valor de la variable número, se incrementa en 1 su valor. Mientras vale menos de 11 salta a la instrucción 20 escribiendo el número y en el momento que toma el valor 11 se acaba el programa al no cumplirse la condición del IF de la instrucción 40.

Como vemos las instrucciones 20 y 30 se ejecutan repetidas veces (concretamente 10). Se llama *bucle* a un

conjunto de instrucciones que se repiten mientras se cumple alguna condición.

En el siguiente apartado veremos como realizar los bucles de una forma cómoda y sencilla.

9.2. El Bucle FOR/NEXT.

Cuando queramos repetir una serie de instrucciones un determinado número de veces y conozcamos a priori este número podemos hacer uso de la instrucción FOR/NEXT.

Sintaxis:

```
 FOR <variable>=<valor inicial> TO <valor final>

---------------------!
                     !
--------------------- > Instrucciones a realizar dentro
                     !  del bucle.
---------------------!

NEXT <variable>
```

La instrucción comienza con la palabra reservada *FOR* seguida de una variable numérica a la que se le da un valor inicial. A continuación aparece la palabra reservada *TO* y por último el valor final al que va a llegar la variable en el bucle. Las instrucciones que se van a repetir son las comprendidas entre las instrucciones *FOR* y *NEXT*, ambas inclusive. Por tanto la instrucción *NEXT* representa la última instrucción del bucle. Cuando la ejecución llega a la instrucción NEXT el valor de la variable se incrementa en una unidad y si es menor o igual al valor final se vuelven a ejecutar las instrucciones que componen el bucle.

Ejemplo.

Volvamos al ejemplo de escribir los diez primeros números. Esta vez lo haremos utilizando la instrucción FOR/NEXT:

```
10 FOR i=1 TO 10
20 PRINT i
30 NEXT i
```

Podemos observar que el programa queda muy simplificado. Le hemos dado a una variable numérica, i, un valor inicial, 1, y un valor final, 10. La variable *i* irá tomando los valores 1,2...10. El último valor, 10, será el último valor que tome en el bucle.

9.3. Opción STEP.

En el apartado anterior hemos visto que la variable se incrementa en una unidad cada vez que entra en el bucle. Puede suceder en ocasiones que interese que la variable se incremente en un valor distinto de 1. Para ello utilizaremos la opción STEP.

Sintaxis:

```
FOR <variable>=<valor inicial> TO <valor final> STEP <incremento>
```

El incremento puede ser un número entero o real.

Ejemplo

Supongamos que sólo queremos escribir, en el ejemplo anterior, los números impares comprendidos entre 1 y 10.

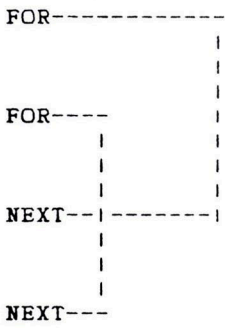
El incremento deberá ser de 2.

```
10 FOR i=1 to 10 STEP 2
20 PRINT i
30 NEXT i
```

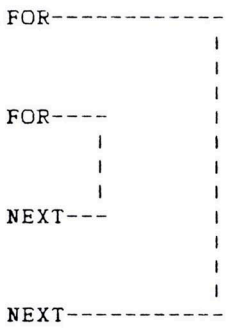
Nota. No es necesario escribir el nombre de la variable después de una instrucción NEXT.

9.4. Bucles Anidados. Reglas.

Se puede introducir un bucle en otro bucle. Pero el bucle interno tiene que acabar antes que el externo. Es decir, que el bucle interno debe estar completamente introducido dentro del externo.



MAL



BIEN

Ejemplo

Programa para realizar una tabla de multiplicar hasta el 9.

En este programa construiremos un bucle para realizar cada tabla y este bucle lo insertaremos en otro para componer las diez tablas.

```
10 FOR i=1 to 9
20 FOR j=1 to 10
30 PRINT i;"*";j;"=";i*j
40 NEXT j
50 PRINT
```

```
60 REM La instrucción PRINT sin parámetros permite
   dejar una línea en blanco entre tabla y tabla.

70 NEXT i
```

Como se puede apreciar, el bucle que comienza en la instrucción 20 y acaba en la 40 está totalmente introducido en el bucle externo que abarca de las líneas 10 a la 70.

9.5. Retardos.

A veces es conveniente que el ordenador esté un período de tiempo sin hacer nada aparentemente. Esta situación se presenta por ejemplo cuando queremos que en pantalla aparezca algo que debe leer la persona que ejecuta el programa y una vez que se considere que ha tenido tiempo suficiente para leer se borre la pantalla y se continúe con la ejecución del programa. Esto lo podemos hacer utilizando la instrucción FOR.

El problema es decir al ordenador que haga algo que no afecte a la pantalla ni a la ejecución del programa y mientras tanto en la pantalla no se aprecie ningún cambio. Para lo cual podemos hacer que el programa ejecute un bucle *vacío*. Es decir, un bucle que no contenga instrucciones sino simplemente que se vaya incrementando la variable hasta llegar a un determinado valor.

Supongamos que en la tabla de multiplicar del apartado anterior queremos que entre la presentación de dos tablas consecutivas haya un retardo de varios segundos. Incluiríamos un retardo quedándonos el programa:

```
10 FOR i=1 to 9

20 FOR j=1 to 10

30 PRINT i;"*";j;"=";i*j

40 NEXT j
```

```
50 PRINT

53 REM retardo

55 FOR k=1 to 1000

57 NEXT k

60 REM La instrucción PRINT sin parámetros permite
   dejar una línea en blanco entre tabla y tabla.

70 NEXT i
```

Obsérvese que hemos vuelto a anidar un nuevo bucle siguiendo las reglas del apartado anterior.

9.6. Bucle Condicional. Instrucción While/Wend.

Cuando no se conoce cuantas veces ha de ejecutarse un bucle, sino que aquellas dependen de alguna condición que se cumpla en la ejecución del programa utilizaremos una nueva instrucción que permite que se ejecute un bucle si se cumple una determinada condición. De no cumplirse se ejecuta la instrucción siguiente al WEND. Es la instrucción WHILE/WEND.

Sintaxis:

```
WHILE <condición>

  ---------------------!
                       !
  ---------------------  > Instrucciones a realizar dentro
                       !   del bucle.
  ---------------------!

WEND
```

En esta estructura sintáctica la instrucción WEND tiene una función similar a la instrucción NEXT en el bucle FOR/NEXT. Representa la última instrucción del bucle.

También se pueden anidar bucles WHILE/WEND siguiendo las reglas ya estudiadas.

Ejemplo.

Hallar la media aritmética de una serie de números que se introducen cuando se ejecute el programa. Cuando se introduzca un 0 el programa escribirá la media de los números introducidos.

Este sería un ejemplo típico de utilización de un bucle WHILE. En principio desconocemos el número de sumandos que vamos a tener pues es un número que se determina cuando se ejecuta el programa.

```
10 REM Media

20 numsum=0

30 REM La variable numsum va a contener el nº de
   sumandos

40 suma=0

50 REM La variable suma contendrá la suma de los
   números introducidos

60 sumando=999

70 REM Proporcionamos a la variable sumando un
   valor ficticio para poder entrar en el bucle
   WHILE la primera vez

80 WHILE sumando<>0

90 REM La primera vez sumando vale 999 y por tanto
   al ser un valor distino de 0 se cumple la condi-
   ción que sigue a la instrucción WHILE.

100 INPUT "Introducir número";sumando

110 suma=suma+sumando

120 IF sumando<>0 then numsum=numsum+1
```

```
130 REM Si el sumando fuese igual a 0 no lo tendríamos en cuenta para calcular la media ya que lo que indicaría sería fin de introducción de datos.
140 WEND
150 PRINT "media=";suma/numsum
160 END
```

ACTIVIDADES

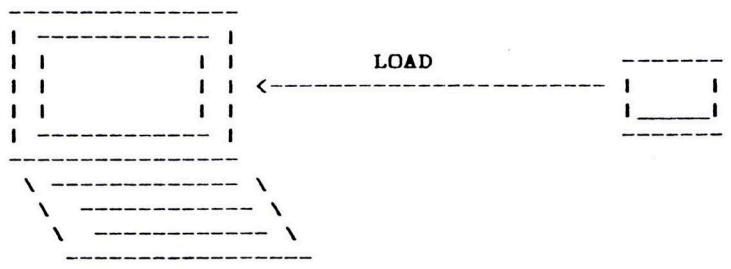
1. Realizar una tabla de multiplicar hasta el 20
2. Escribir los múltiplos de 5 menores que 100.
3. Escribir 15 veces la frase "Aprendo a programar"

10. Manejando el Disco.

La memoria central del ordenador es una memoria *volátil*, es decir que se pierde su información cuando se desconecta el ordenador. Si queremos conservar nuestros programas deberemos grabarlos en disco.

10.1. Cargando Programas. Comando LOAD.

El comando LOAD carga un programa, grabado en disco, en memoria. Hay que tener en cuenta que para que un programa pueda ejecutarse debe estar presente en la memoria central.



Sintaxis:

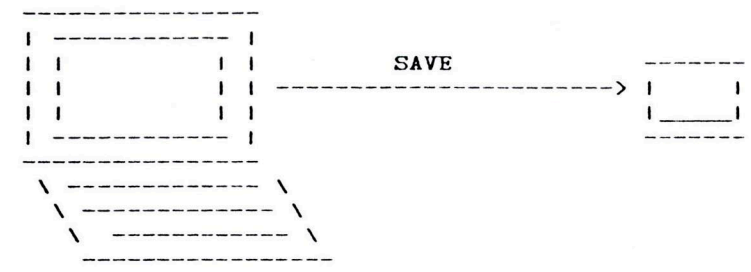
LOAD "<nombre del programa>"

Si queremos, además de cargar el programa, ejecutarlo escribiremos RUN como ya se vió en el capítulo 4. Sin embargo cuando queramos cargar un programa y ejecutarlo hay un método más abreviado que consiste en escribir

RUN "<nombre del programa>"

10.2. Grabando Programas. Comando SAVE.

El comando SAVE es dual del anterior. Permite la grabación en disco de un programa que tengamos en memoria.



Sintaxis:

SAVE "<nombre del programa>"

El nombre del programa debe constar de un máximo de 8 caracteres.

10.3. Borrar Programas: Comando ERA.

Para borrar un programa de disco escribiremos:

era <nombre del fichero>

El nombre del fichero debe incluir la extensión. Los programs en BASIC llevan la extensión .BAS

Si a continuación listamos el directorio, con el comando DIR, veremos que el programa ya no se encuentra en el disco.

10.4. La instrucción KILL.

La instrucción KILL opera como la instrucción ERA. Pero tiene la ventaja de que puede borrar ficheros cuyos nombres sean el contenido de *variables* alfanuméricas, mientras que la instrucción ERA exige que se especifique en forma concreta el nombre del fichero.

Ejemplo

```
10 INPUT "Introducir el nombre del fichero que se quiere borrar";a$

20 KILL a$
```

El programa borra el nombre del fichero que se haya introducido en la variable a$ mediante la instrucción INPUT.

11. VARIABLES DIMENSIONADAS o ARRAYS

Hay veces que se amontonan los datos que intervienen en un programa y es necesario estructurarlos en unidades superiores que permitan realizar con ellos su debido tratamiento. El objeto de este capítulo es ver cómo se pueden estructurar los datos en forma de arrays y cómo se puede trabajar con los arrays.

11.1. Concepto de Array.

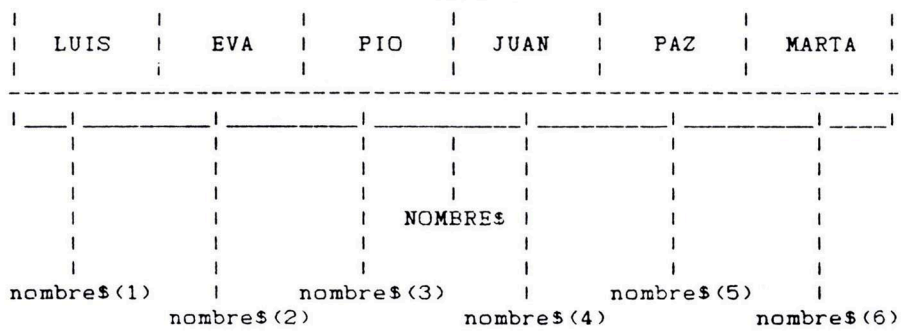
Imaginemos que queremos almacenar en memoria los nombres de los jugadores de un equipo de futbol. Podemos utilizar 11 variables distintas, una por jugador, lo cual es poco cómodo. Más tedioso sería aún el tener que almacenar de esta manera información sobre los 500 alumnos de un colegio. En BASIC disponemos de un recurso para almacenar en una única variable información referida a muchos elementos. Son las *variables dimensionadas o arrays*.

Las variables dimensionadas tienen un índice que permite saber a qué elemento, de los que contiene la variable, nos estamos refiriendo.

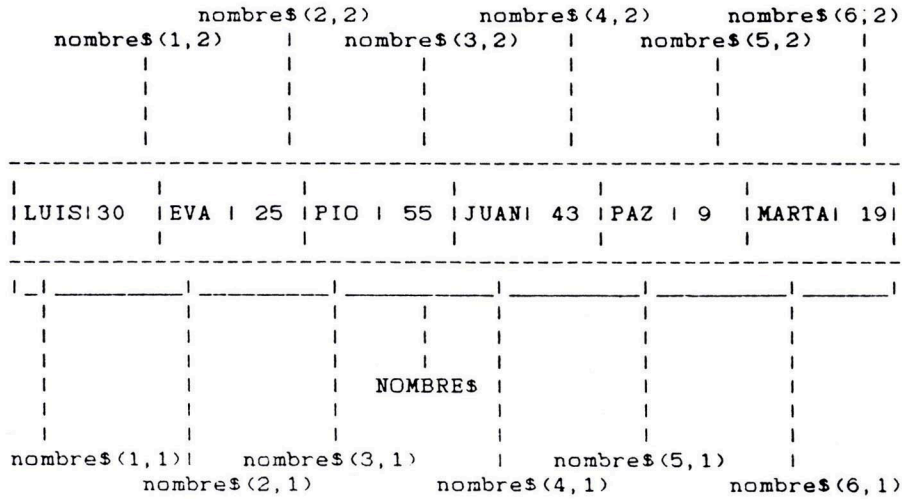
Si queremos guardar una única información referida a cada elemento del array tendremos arrays unidimensionales. En estos arrays el índice es un único número. (Ejemplo de los nombres de los 11 jugadores de un equipo de fútbol. Sólo nos interesan los nombres.) En cambio si queremos almacenar más de una información por cada elemento utilizaremos 2 índices, uno indicativo del número de elemento y el otro nos señala a que información hacemos referencia de las relacionadas con dicho elemento. (Podemos suponer que aparte de los nombres de los jugadores nos interese conocer su edad y altura)

Las variables dimensionadas, igual que las variables sencillas, pueden ser numéricas o alfanuméricas dependiendo del tipo de información en ellas almacenada.

A continuación vamos a ver una representación interna a nivel lógico de un ejemplo de array unidimensional y de otro array de tipo multidimensional.



En este array unidimensional están comprendidos seis nombres de personas. La variable que contiene los seis nombres se llama nombre$. Cada uno de los elementos viene referenciado por el nombre de la variable, nombre$, y por un índice entre paréntesis que representa el número de elemento dentro de la variable. Así cuando hagamos referencia a Pío escribiremos nombre$(3).



En el ejemplo anterior tenemos una variable *nombre* que contiene seis elementos y cada uno de estos elementos está afectado de dos informaciones, o *campos*. El primer campo de cada elemento se refiere al nombre y el segundo a la edad. De esta manera, si se quiere hacer algún cálculo con la edad de PAZ se trabajará con la variable nombre$(5,2).

11.2. Dimensionar Tablas: Instrucción DIM

Antes de trabajar con una variable dimensionada hay que poner una instrucción que indique el número de elementos que contenga la variable. Esta es la instrucción DIM.

Sintaxis:

DIM <nombre de variable> (<nº de elementos>)

La representación del número de elementos será un número en el caso de los arrays unidimensionales. Si el array es de dos dimensiones, la variable vendrá subindicada por dos dígitos. El primero será el número de elementos. El segundo, el número de campos de cada elemento.

En el ejemplo anterior se podía haber escrito para dimensionar la tabla de nombres y edades:

DIM persona$ (6,2)

donde:

persona$: Nombre de la variable dimensionada.

6: Número de elementos de la tabla (6 personas)

2: Número de campos por elemento (el campo nombre y el campo edad).

11.3. Llenar Tablas.

Como hemos visto un array es una estructura de datos. En este apartado se estudia la forma de dar contenido a un array.

La forma más sencilla de llenar una tabla es ir asignando a cada campo de cada elemento un valor mediante la instrucción LET (o su equivalente implícita, es decir, no escribir la palabra LET).

En nuestro ejemplo para introducir los nombres y las edades en una variable dimensionada deberíamos haber procedido de la siguiente manera:

```
10  persona$(6,2)
20 persona$(1,1)="LUIS"
30 persona$(1,2)="30"
40 persona$(2,1)="EVA"
50 persona$(2,2)="25"
60 persona$(3,1)="PIO"
70 persona$(3,2)="55"
80 persona$(4,1)="JUAN"
90 persona$(4,2)="43"
100 persona$(5,1)="PAZ"
110 persona$(5,2)="9"
120 persona$(6,1)="MARTA"
130 persona$(6,2)="19"
```

Otra forma de haber rellenado la tabla podía haber sido mediante la instrucción INPUT, si bien con esta instrucción

habria que introducir los nombres y las edades cada vez que se ejecutase el programa.

Con la instrucción INPUT trabajariamos de la siguiente forma:

```
10 DIM persona$(6,2)
20 FOR i=1 to 6
30 INPUT "Introducir nombre";persona$(i,1)
40   INPUT "Introducir edad";persona$(i,2)
50 NEXT i
```

Obsérvese la comodidad de trabajar con bucles cuando se rellena una tabla con la instrucción INPUT. Para ello se hace corresponder el índice de la tabla con el índice de la instrucción FOR con una única variable (la variable i).

En general, y al igual que ocurría con las variables sencillas, se utilizará la instrucción LET cuando los datos de entrada *no varíen* y trabajaremos con la instrucción INPUT cuando cada vez que se ejecute el programa haya que introducir *datos distintos*.

11.4. La Pareja READ/DATA y las TABLAS.

Hasta el momento hemos visto dos formas posibles de asignación de valores a variables: la instrucción LET y la instrucción INPUT. El lenguaje BASIC nos ofrece un tercer método de realizar asignaciones. Para ello se almacenan los datos en algunas instrucciones y se van haciendo uso de ellos a lo largo del programa.

La instrucción que sirve para almacenar datos es la instrucción DATA que tiene el siguiente formato:

```
DATA <lista de datos>
```

Los datos pueden ser valores numéricos o alfanuméricos y van separados unos de otros mediante comas.

Estos valores son asignados a variables mediante la instrucción READ cuyo formato es el siguiente:

```
READ <lista de variables>
```

La instrucción READ toma un dato guardado en una instrucción DATA y se lo asigna a una variable de su lista. El siguiente dato que toma lo introduce en la siguiente variable y así sucesivamente. Puede haber más datos en una instrucción DATA que variables en una una instrucción READ en espera que otras instrucciones READ, o la misma si está dentro de un bucle, tome los datos que aún no se hayan utilizado. Sin embargo no puede haber más variables en una lista de variables que datos en una lista de datos (a no ser que hubiese más instrucciones DATA a lo largo del programa).

La lista de variables en varias instrucciones READ y en instrucciones DATA tiene un efecto aditivo. Es decir que es equivalente escribir:

```
READ a$,b,c
```

a poner:

```
READ a$
READ b,c
```

Del mismo modo, en lugar de escribir

```
DATA "Madrid",5,7
```

se pueden introducir las siguentes instrucciones:

```
DATA "Madrid"
DATA 5
DATA 7
```

La instrucción DATA puede encontrarse en cualquier parte del programa. No es una instrucción ejecutable y el programa se la saltará como si no existiese. Su único fin es *almacenar* datos que vayan a ser utilizados por instrucciones READ.

Es muy útil la aplicación de la instrucción READ/DATA para la introducción de los datos en una variable dimensionada. Volvamos nuevamente al ejemplo de los nombres y las edades. Con la instrucción READ/DATA operaríamos de la siguiente manera:

```
10 DIM persona$(6,2)
20 FOR i=1 TO 6
30  FOR j=1 to 2
40  READ persona$(i,j)
50  NEXT j
60 NEXT i
...
10000 DATA "Luis","30"
10010 DATA "Eva","25"
10020 DATA "Pio","55"
10030 DATA "Juan","43"
10040 DATA "Paz","9"
10050 DATA "Marta","19"
```

En este ejemplo es preciso observar:

La *utilidad de los bucles anidados* cuando se trabaja con instrucciones READ/DATA.

Las instrucciones DATA se puede encontrar *en cualquier parte* del programa. Podemos llevarnos los datos al final del programa y despreocuparnos de ellos cuando estamos elaborando el programa.

El *orden preciso* de los datos en la instrucción DATA que deben corresponderse con la lista de variables de la instrucción READ. Si se entremezclasen valores numéricos y alfanuméricos en una instrucción DATA y a una variable numérica le correspondiese una cadena de caracteres habríamos cometido un error.

Se podían haber *sustituído* varias DATAS por una sola que almacenase los datos de todas ellas.

En general se utilizará la instrucción READ/DATA en todos aquellos casos en que se trabaje con muchos datos que no varíen y si lo hacen, muy esporádicamente (Por ejemplo los nombres de los empleados de una empresa, sus direcciones, fecha de nacimiento etc.)

11.5. Instrucción RESTORE.

A veces es necesario volver a utilizar los datos de una instrucción DATA que ya hayan sido leídos en una etapa anterior de la ejecución de un programa. Es el caso de varias variables que pueden compartir un dato almacenado en una instrucción DATA. Para volver a utilizar los datos que ya han sido tomados por instrucciones READ se utiliza la instrucción RESTORE cuyo formato es:

```
RESTORE
```

Después de ejecutarse una instrucción RESTORE, cuando el programa se encuentre con una instrucción READ, se tomará el primer dato de la primera instrucción DATA del programa para que sea asignado a la primera variable de la lista de variables de la instrucción READ. Es decir se habrán

restaurado los datos y habrán quedado como al principio de la ejecución del programa.

.....

Como ejemplo de lo estudiado en este capítulo vamos a realizar un programa que simule un diccionario. La estructura de datos será una tabla en la que cada elemento tendrá dos campos: la palabra y su significado.

Los datos vendrán dados en instrucciones DATA.

El programa se dividirá en dos fases: Una primera fase de introducción de datos en la tabla y una segunda de consulta.

Para saber cuál es el último valor de la instrucción DATA (y dejar de leer en cuanto lleguemos a él) nos serviremos de un pequeño "truco". Vamos a escribir como último par de datos una palabra y un significado ficticios, por ejemplo *zzz,zzz*. Cuando el programa detecte que ha llegado a ese valor dejará de seguir leyendo, esto es, de seguir buscando valores en instrucciones DATA.

```
10 REM diccionario

20 CLS$=CHR$(27)+"E"+CHR$(27)+"H"

30 DIM diccio$(2000,2)

40 REM Podemos introducir un máximo de 2000
       palabras

50 i=1

60 REM La variable i va a ser el índice de los
       elementos de la tabla

70 fin$="no"

80 REM La variable fin$ dejará de valer "no" cuando
       se lea el último valor
90 WHILE fin$="no"

100    FOR j=1 to 2
```

```
110     REM La variable j indica el campo:el primer
            campo es el nombre y el segundo el
            significado

120     READ diccio$(i,j)

130   NEXT j

140   if diccio$(i,1)="zzz" then fin$="si" else
      i=i+1: REM La variable i va indicando por
      qué número de elemento va. Al final el valor
      de i será el número de pares de palabras
introducidas.

150 WEND

160 REM Los datos ya están introducidos en la
        variable  diccio$. Comienza la fase de
        consulta

170 PRINT CLS$

180 INPUT "Que palabra desea buscar";pal$

190 FOR k=1 to i

200 IF pal$=diccio$(k,1) then goto 220: REM Nos
    salimos del bucle FOR y dejamos de buscar. La
    palabra buscada es la que hace el número k

210 NEXT k

220 PRINT "El significado es ";diccio$(k,2)

230 INPUT "Desea buscar otra palabra (si-no)";resp$

240 IF resp$="si" then goto 170 else end

1000 REM Comienza la zona de datos

1010 DATA "computer", "ordenador"

1020 DATA "screen", "pantalla"

1030 DATA "clear", "limpiar"
```

```
1040 DATA "dog","perro"

1050 REM Aquí podemos seguir introduciendo datos
     hasta 2000 como máximo

50000 DATA "zzz","zzz"
```

En el programa anterior es preciso observar los siguientes puntos.

-El bucle FOR anidado dentro del bucle WHILE. Siguiendo las reglas de los bucles aparece antes la instrucción NEXT, cerrando el bucle FOR, que la instrucción WEND, que cierra el bucle WHILE.

-La búsqueda secuencial por todos los elementos de la tabla. En el momento que coincide la palabra buscada con alguna de las que tenemos en la tabla se deja de buscar, saliéndonos del bucle, y se recuerda el número de elemento que hace dentro de la tabla, en nuestro caso, con la variable k. A continuación se escribe como solución el segundo campo del número que hace dicho elemento dentro de la tabla.

-La utilización de la variable fin$ como switch (variable que sólo puede tomar dos valores, en el ejemplo "si" y "no").

ACTIVIDADES

1. Reformar el programa anterior para que sea un traductor doble: Inglés-Castellano, Castellano-Inglés.

2. Realizar un programa para almacenar nombres, direcciones y teléfonos de distintas personas para que se pueda acceder a los datos de cada una de ellas.

12. El Uso de Subrutinas

12.1. Concepto y Utilidad.

Cuando se realiza un programa largo y complejo se suelen cometer frecuentes errores, algunos de ellos difíciles de encontrar. El número de variables que intervienen se hacen excesivamente numerosas y el programador pierde el control del programa que está creando. Es en estos casos cuando se hace inevitable una estructuración del programa en partes o *módulos*, que solucionan parcialmente el problema. El conjunto de todos los módulos signica la resolución total del programa que se está realizando. Estos módulos, al formar parte de los programas, son más sencillos de realizar que si se elabora el programa de una sola vez. Por otra parte si surge un error sólo habrá que tratar de localizarlo en la subrutina que falle. Una última ventaja se produce cuando hay que repetir una serie de instrucciones en distintas partes del programa. Con los subprogramas sólo es necesario escribir una única vez las instrucciones que se repiten .

A estos módulos se les llama *subprogramas* o *subrutinas*. Podemos definir un subprograma como un conjunto de instrucciones que resuelve una parte de un problema y que se encadena con el resto del programa formando una estructura compacta. En el uso de un subprograma hay que distinguir:

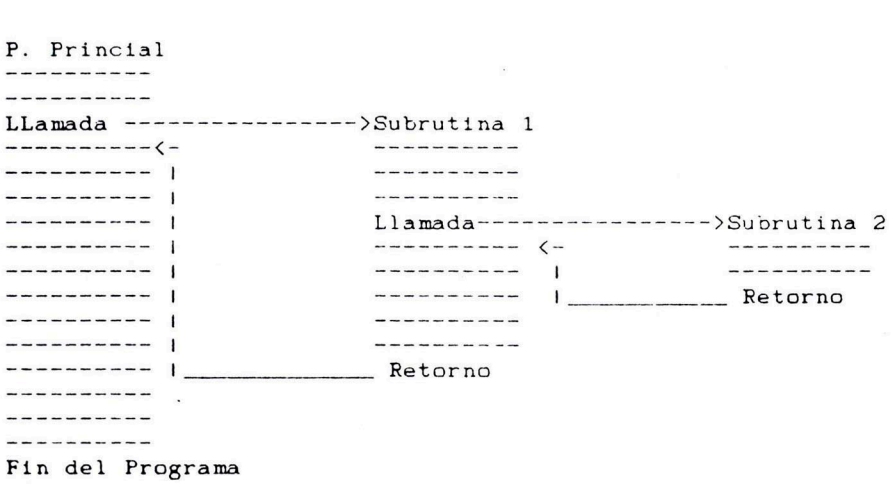
-Llamada al subprograma: Un programa al llegar a un determinado punto de su ejecución necesita recurrir a los servicios que le presta el subprograma, entonces lo *llama* que es tanto como decir que el control del programa pasa en este momento al subprograma.

-Retorno del Subprograma: El Subprograma, una vez que le ceden el control, empieza a ejecutar sus instrucciones y cuando ha terminado devuelve el control de la ejecución del programa al programa que lo llamó, continuando éste la ejecución en la instrucción siguiente a la llamada al subprograma.

Programa Principal: Es el que realiza la llamada.

Programa Secundario, Subrutina o Subprograma: Es el programa llamado. Un Subprograma puede llamar a su vez a otro subprograma comportándose aquel como programa principal con respecto a éste.

Esquemáticamente el comportamiento descrito es el siguiente:



En este gráfico se observa que un programa, en un momento dado, cede el control a una subrutina, ésta comienza a ejecutarse llamando al llegar a una determinada instrucción a una segunda subrutina. Una vez que se ha terminado de ejecutar la segunda subrutina se devuelve el control a la rutina que la llamó y no al programa principal. Después de ejecutarse la subrutina 1 se devuelve control al programa principal que se continúa ejecutando en la instrucción siguiente a la que sirvió para llamar a la subrutina.

12.2. El Par GOSUB/RETURN

En Basic las llamadas a las subrutinas se realizan mediante la instrucción GOSUB que tiene el siguiente formato:

GOSUB n

donde n es el nº de instrucción donde comienza la subrutina llamada.

El retorno al programa principal (o a la subrutina que llama) se realiza mediante la instrucción RETURN cuyo formato es:

RETURN

Como ejemplo de este capítulo vamos a realizar un programa que sume, reste, multiplique y divida. Cada una de estas operaciones la realizará una subrutina diferente.

La Subrutina Sumar va a empezar en la instrucción 210.

La Subrutina Restar comenzará en la instrucción 250.

La Subrutina Multiplicar tendrá su inicio en la instrucción 290.

Por último, la subrutina Dividir se realizará a partir de la instrucción 330.

```
10 REM Operaciones
20 REM Programa Principal
30 cls$=CHR$(27)+"E"+CHR$(27)+"H":PRINT cls$
40 INPUT "Introducir primer operando";op1
50 INPUT "Introducir segundo operando";op2
60 PRINT cls$
```

```
70 PRINT "1. SUMA":PRINT
80 PRINT "2. RESTA":PRINT
90 PRINT "3. PRODUCTO":PRINT
100 PRINT "4. DIVISION":PRINT
110 PRINT "5=FIN":PRINT
120 INPUT "Pulsar opcion";opcion
130 PRINT cls$
140 IF OPCION=1 THEN GOSUB 210
150 IF OPCION=2 THEN GOSUB 250
160 IF OPCION=3 THEN GOSUB 290
170 IF OPCION=4 THEN GOSUB 330
180 IF OPCION=5 then END
190 FOR i=1 to 3000: NEXT i :REM **Bucle de retardo
200 GOTO 30: REM El programa principal vuelve al
    menu
210 REM SUBRUTINA SUMA
220 suma=op1+op2
230 PRINT "La suma es";suma
240 RETURN
250 REM SUBRUTINA RESTA
260 resta=op1-op2
270 PRINT "La resta es";resta
280 RETURN
290 REM SUBRUTINA PRODUCTO
```

```
300 prod=op1*op2
310 PRINT "El producto es";prod
320 RETURN
330 REM SUBRUTINA COCIENTE
340 coci=op1/op2
350 PRINT "El cociente es";coci
360 RETURN
```

En el programa anterior se debe observar:

-Que en las subrutinas sólo se entra desde las llamadas del programa principal (líneas 140-170). De lo contrario daría un mensaje de error por encontrarse una instrucción RETURN inesperada.

-Que el bucle de retardo (línea 190) permite que durante unos segundos aparezca el resultado en la pantalla. Si se quitase esta instrucción no daría tiempo a ver el resultado pues se borraría al saltar el programa desde la instruccion 200 a la instrucción 30.

ACTIVIDADES

1. Reformar el programa anterior para poder realizar también la exponenciación.

2. Realizar un programa en el que se introduzca un número y a partir de él, utilizándolo como radio, se calcule la longitud de la circunferencia, el área del círculo y el volumen de la esfera correspondiente a ese radio. Cada uno de los cálculos se realizará en una subrutina distinta.

13. FICHEROS SECUENCIALES

13.1. Introducción a los ficheros

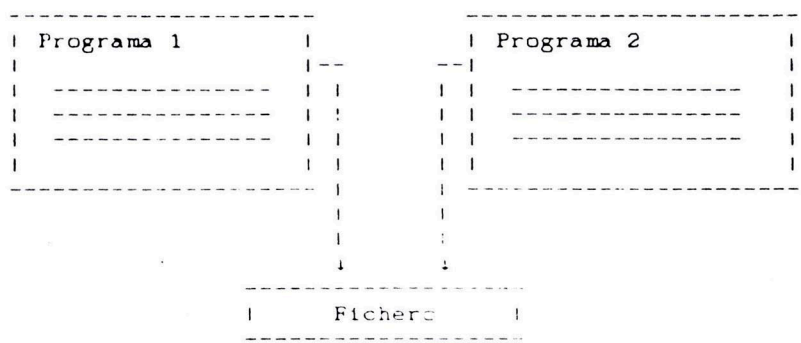
Según se ha estudiado, la instrucción DATA permite almacenar datos que se van a utilizar en la ejecución del programa. Cuando el número de datos almacenados es alto, es preferible separar los datos del programa, grabando los datos aisladamente en el disco. Esto se realiza mediante la utilización de *ficheros*. Un fichero es una colección de datos grabada en un disco.

Las ventajas que ofrece la utilización de ficheros son las siguientes:

Mayor claridad: Al ser unidades independientes el programa y los datos, éstos no interfieren en la codificación de instrucciones.

Mayor capacidad de almacenamiento: La capacidad de un fichero viene determinada por la capacidad del disquete (169K) y no de la memoria central.

Posibilidad de *compartir* un fichero entre varios programas. Por ejemplo, un fichero de empleados de una empresa puede ser utilizado por un programa que escriba el nombre de los empleados de la empresa y por otro que extraiga la nómina de cada uno de ellos.



13.2. Ficheros, Registros y Campos.

Se llama registro *al conjunto de información referida a un elemento.* Si quisiéramos saber los datos de una persona deberíamos guardar su nombre, dirección, teléfono, edad, etc. Este conjunto de información es lo que sería un registro. El conjunto de informaciones referidas a esta persona hace que la identifiquemos con unas características propias. Se dice por ello que el registro es una unidad de información con sentido propio. Los componentes del registro como son la edad, teléfono, etc. no aportan por sí solas información y únicamente tienen sentido cuando se unen formando el registro. A estos elementos de los registros se les llama *campos*.

Si englobamos un conjunto de registros tendremos una unidad mayor que se llama *fichero*. Los ficheros se utilizan ampliamente porque permiten agrupar en una única estructura de información gran cantidad de datos. Podemos tener ficheros de empleados de una empresa, de alumnos de un colegio, de resultados de una liga de futbol, etc.

13.3 Operaciones con ficheros secuenciales

13.3.1 Creación de un fichero secuencial

Los ficheros se crean para almacenar posteriormente información en ellos. Para crear un fichero secuencial en BASIC MALLARD se utiliza la instrucción OPEN con la siguiente sintáxis:

OPEN "O", < Nº del fichero >, < Nombre del fichero>

OPEN : Es una instrucción que abre un fichero, es decir, que lo deja disponible para leer o escribir en él. Cuando se abre para escribir datos en él lo que se produce es un creación del fichero en el disco, borrando cualquier otro fichero que hubiese grabado en el disco con el mismo nombre.

"O": Indica que el fichero que se va a abrir es de salida (output), lo que equivale a crear un nuevo fichero en disco.

Nº de fichero: Es el identificativo, dentro del programa del fichero que se crea. Puede ser un número entero en el margen 1-3. Esto indica que en un momento dado puede

haber un máximo de tres ficheros abiertos. (Aunque este tope es modificable con la instrucción MEMORY que se estudiará más adelante.)

Nombre del fichero: Es el identificativo del fichero para el sistema operativo. (El nombre que aparecerá en el directorio).

Ejemplo:

```
10 OPEN "O", 1, "PERSONAS"
```

La instrucción anterior crea un fichero en disco con el nombre PERSONAS. Cuando se haga referencia a este fichero en el programa se utilizará el nº 1 como identificativo del fichero.

13.3.2. Escritura de datos en el fichero.

Para escribir datos en un fichero secuencial se utiliza la instrucción WRITE con el siguiente formato:

```
WRITE # < Nº de fichero>, < lista de datos >
```

Como se puede observar no se utiliza el nombre del fichero, sino su número como identificativo para trabajar con él dentro del programa.

Ejemplo:

```
20 WRITE # 1, "Fernando", "Pez 1", 4355667
```

La instrucción anterior grabará en el fichero número 1 la lista de datos "Fernando", "Pez 1" y 4355667.

Si se utiliza posteriormente otra instrucción WRITE # 1, la nueva lista de datos se graba a continuación del último dato grabado, es decir, después de 4355667.

13.3.3. Cierre del Fichero.

Cuando no se van a introducir más datos en un fichero hay que cerrarlo. Cerrando un fichero quedan grabados definitivamente en disco los datos escritos en el fichero.

Para cerrar un fichero se utiliza la instrucción CLOSE con el siguiente formato:

CLOSE < lista de número de ficheros >

Los números de los ficheros que se cierran, se pueden utilizar para cerrar posteriormente ficheros con esos números.

Ejemplo:

```
CLOSE 1
```

El fichero que tiene por número 1 se cierra y queda grabado en disco. A partir de este momento podemos crear un nuevo fichero y darle como número, 1.

La instrucción CLOSE sin parámetros provoca que se cierren todos los ficheros que estuvieran abiertos.

13.3.4. Apertura de ficheros en lectura.

Para leer un fichero que ha sido previamente grabado y acceder a sus datos, la primera operación que se debe efectuar es abrirlo.

Para abrir un fichero con objeto de leerlo se utiliza la instrucción OPEN con el siguiente formato:

OPEN "I",<nº de fichero>,<nombre de fichero>

Donde la "I" indica que va a ser un fichero del que se van a leer datos, es decir, va a ser un fichero de entrada (input). El resto de parámetros han sido explicados anteriormente.

Para adquirir datos del fichero se escribe la instrucción:

INPUT#n,<lista de variables>

donde "n" indica el nº de fichero del que se va a leer.

Donde los datos que se van tomando del disco se alojan en las variables de la lista.

13.3.5. Cierre de un fichero en lectura.

Una vez que se han leído los datos que contiene un fichero hay que cerrarlo. El cierre de un fichero de lectura se efectúa con la instrucción:

CLOSE <nº de fichero>

Como se puede observar la operación es similar al cierre en escritura.

13.3.6. La función FIND$

La función FIND$ genera la cadena vacía si un determinado fichero no se encuentra en el disco. Si se halla en el disco genera el nombre del fichero. Es una función útil para detectar errores en la manipulación del disco. (Intentar leer ficheros inexistentes). Su formato es el siguiente:

FIND$(nombre del fichero)

Ejemplo

```
10 INPUT "Introducir el nombre del fichero a leer";n$
20 IF FIND$(n$)="" THEN PRINT "Introducir el disco donde
se encuentra el fichero y después pulsar una tecla"
30 WHILE INKEY$="":WEND
```

13.3.7. Ejemplo de utilización de ficheros secuenciales

Se pretende realizar un programa en el cual se puedan almacenar los nombres de los clientes de un determinado vendedor así como la cuantía a la que ascienden las ventas de cada uno a lo largo de un mes en un fichero. Cada vez que

comienza un mes, se deberá poner la cuenta de los gastos de cada cliente a cero.

El programa deberá permitir dar altas, bajas y modificaciones. También se podrá sacar un listado por pantalla o por impresora de los clientes y de las cuantías de sus gastos.

Solución

Lo primero que se debe realizar cuando se plantea un problema es el análisis del mismo. Para ello se divide el problema en distintas partes que serán solucionadas mediante sendas subrutinas.

En el programa principal se define el tamaño de la tabla donde se alojarán los datos cuando se encuentren en memoria. También incluirá el programa principal un menú de opciones con las distintas facilidades que proporciona el programa.

Se utilizará una subrutina que lee un fichero ya creado con los datos de cada cliente y deja los datos en una tabla en memoria. (A esta Subrutina se accederá desde el programa principal siempre y cuando exista el fichero. De no estar el fichero en el disco se crea).

Cada una de las opciones del menú será resuelta por una subrutina:

Altas: Permite introducir el nombre de un nuevo cliente. El máximo número de clientes queda establecido en 2000.

Bajas: Elimina los datos de un cliente del fichero.

Modificaciones: Sirve para cambiar el nombre de un cliente o bien su cantidad asociada.

Consultas: Se introduce un nombre y aparece la cantidad asociada a esa persona.

Introducir Ventas: Mediante esta opción se introduce la cuantía de una nueva operación debiéndose incrementar automáticamente.

Inicializar mes: Con esta opción se pone la cuantía de

los gastos de cada cliente a cero.

Listado: El listado de clientes podrá ser realizado por pantalla o por impresora.

Fin: La opción FIN provoca que el programa concluya actualizando el fichero.

Las subrutinas del menú de opciones son accesibles desde el programa principal. (Corresponderían a un segundo nivel).

Otras subrutinas que deberemos utilizar y que son accesibles desde las subrutinas del menú principal (por tanto corresponden a un tercer nivel de estructuración) son las siguientes:

Confirmación de los datos introducidos: Antes de almacenar los datos en la tabla pedimos confirmación. (Esta subrutina comenzará a partir de la instrucción 51000).

Localización de un registro: La subrutina que localiza un registro en el fichero secuencial va comparando el nombre de la persona a localizar con cada uno de los nombres almacenados en el fichero. (Esta subrutina comenzará a partir de la instrucción 52000).

Retardo: Esta subrutina provoca un retardo. (Comienza a partir de la instrucción 53000).

Pulsar tecla: Esta subrutina supone una espera inactiva mientras no se teclea nada. (Inicio a partir de la instrucción 54000).

Escribir título: Esta subrutina escribe la opción después de limpiar la pantalla. (Empieza a partir de la instrucción 55000)

Nombre inexistente: Mediante esta subrutina se informa que el nombre no existe y suena un pitido. (Esta subrutina se sitúa a partir de la instrucción 58000).

Listado por pantalla: Permite ver la información almacenada en el fichero por pantalla. (Comienza a partir de la instrucción 56000).

Listado por impresora: Aparecerá en forma impresa la información del fichero. (El principio de la subrutina está en la línea 57000).

Para la presentación de los resultados se utiliza la claúsula TAB que permite escribir los resultados a partir de la columna que se especifique. También se utilizará la claúsula USING que permite establecer un formato numérico. En nuestro ejemplo, dejaremos siete dígitos enteros y dos decimales.

```
10 REM CLIENTES
20 REM Ejemplo de utilizacion de Ficheros Secuenciales
30 REM By F.Javier Alvarez
40 REM Octubre/86
50 DIM tabla$(2000,2)
60 REM Definicion de borrado de pantalla y ubicacion del cursor
*******************
70 cls$=CHR$(27)+"E"+CHR$(27)+"H"
80 DEF FN locate$(y,x)=CHR$(27)+"Y"+CHR$(32+y)+CHR$(32+x)
90 PRINT cls$
100 IF FIND$("ficlient")<>"" THEN GOSUB 50000
110 REM
120 REM
130 REM MENU ************************************
140 PRINT CLS$
150 PRINT FN LOCATE$ (3,10);"1. ALTAS"
160 PRINT FN LOCATE$ (6,10);"2. BAJAS"
170 PRINT FN LOCATE$ (9,10);"3. MODIFICACIONES"
180 PRINT FN LOCATE$ (12,10);"4. CONSULTAS"
190 PRINT FN LOCATE$ (15,10);"5. INTRODUCIR VENTAS"
200 PRINT FN LOCATE$ (18,10); "6. INICIALIZAR MES"
210 PRINT FN LOCATE$ (21,10);"7. LISTADO DE CLIENTES"
220 PRINT FN locate$ (24,10);"8. FIN"
230 PRINT FN LOCATE$ (38,25);"Pulsar opcion ";
240 a$=INKEY$:IF a$="" THEN GOTO 240
250 op=VAL(a$)
260 ON op  GOSUB 2000,4000,6000,8000,10000,12000,14000,16000
270 GOTO 130
2000 REM ALTAS*************************************
2010 np=np+1  '
2020 IF np=2000 THEN PRINT CLS$:np=np-1:PRINT "Demasiados
clientes":GOSUB 53000: RETURN
2030 GOSUB 55000:'TITULO
2040 REM Escribe Altas
```

```
2050 PRINT FN LOCATE$ (5,10);:INPUT "Introducir
nombre";tabla$(np,1)
2060 PRINT FN LOCATE$ ( 10,10);:INPUT "Introducir
Cuantia";tabla$(np,2)
2070 GOSUB 51000:REM confirmacion
2080 IF con$="si" THEN RETURN ELSE np=np-1:RETURN
4000 REM BAJAS ****************************************
4010 PRINT CLS$
4020 OP$="BAJAS"
4030 GOSUB 55000:'TITULO
4040 GOSUB 52000:'LOCALIZACION
4050 IF enc$="no" THEN PRINT CLS$:GOSUB 58000:GOSUB
53000:RETURN:REM no hallado
4060 GOSUB 51000
4070 IF con$="no" THEN RETURN
4080 FOR j=i+1 TO np
4090   tabla$(j-1,1)=tabla$(j,1)
4100   tabla$(j-1,2)=tabla$(j-1,2)
4110 NEXT j
4120 np=np-1
4130 RETURN
6000 REM MODIFICACIONES ***********************************
6010 OP$="MODIFICACIONES"
6020 PRINT CLS$
6030 GOSUB 52000 :REM LOCAIZACION DE UN REGISTRO
6040 IF enc$="no" THEN PRINT CLS$:GOSUB 58000:GOSUB 53000:RETURN
:REM no hallado
6050 PRINT CLS$
6060 PRINT FN LOCATE$ (5,5);"Modificacion de nombre (S/N)?";
6070 a$=INKEY$:IF a$="" THEN 6070
6080 a$=UPPER$(a$)
6090 IF a$="S" THEN PRINT FN LOCATE$ (10,5);:LINE INPUT "Nuevo
nombre ?";nn$ ELSE nn$=tabla$(i,1)
6100 PRINT FN LOCATE$ (15,5);:PRINT "Modificacion de cuantia
(S/N)";
6110 a$=INKEY$:IF a$="" THEN 6110
6120 a$=UPPER$(a$)
6130 IF a$="S" THEN PRINT FN LOCATE$ (20,5); : LINE INPUT "Nueva
cuantia ?";nc$ ELSE nc$=TABLA$(i,2)
6140 GOSUB 51000:'confirmacion
6150 IF con$="no" THEN RETURN
6160 tabla$(i,1)=nn$
6170 tabla$(i,2)=nc$
6180 RETURN
8000 REM CONSULTAS ************************************
8010 op$="CONSULTAS"
8020 GOSUB 55000:REM titulo
```

```
8030 GOSUB 52000:REM localizacion
8040 IF enc$="no" THEN PRINT CLS$:GOSUB 58000:GOSUB
53000:RETURN:REM no hallado
8050 PRINT CLS$
8060 PRINT FN LOCATE$(12,5);:PRINT"Nombre: ";UPPER$(tabla$(i,1))
8070 PRINT FN LOCATE$ (15,5);:PRINT"Cuantia: ";tabla$(i,2)
8080 GOSUB 54000
8090 RETURN
10000 REM Introducir Ventas ***************************
10010 op$="VENTA"
10020 GOSUB 55000:REM TITULO
10030 GOSUB 52000:REM Localizacion
10040 IF enc$="no" THEN PRINT CLS$:GOSUB 58000:GOSUB
53000:RETURN:REM no hallado
10050 PRINT CLS$
10060 PRINT  FN LOCATE$(12,5);:PRINT"Nombre: ";
UPPER$(tabla$(i,1))
10070 PRINT FN LOCATE$ (15,5);:INPUT "Cuantia de la nueva venta:
"; nv
10080 suma=VAL(tabla$(i,2))+nv
10090 GOSUB 51000
10100 IF con$="no" THEN RETURN
10110 tabla$(i,2)=STR$(suma)
10120 RETURN
12000 REM INICIALIZAR MES *****************************
12010 PRINT CLS$
12020 PRINT FN LOCATE$ (10,5);:PRINT"ATENCION. Al inicializar el
mes se restaura la cantidad asignada a cada cliente"
12030 PRINT FN LOCATE$ (15,5);:PRINT"Confirmacion (S/N)
";CHR$(143)
12040 a$=INKEY$:IF a$="" THEN 12040
12050 a$=UPPER$(a$)
12060 IF a$="N" THEN RETURN
12070 FOR i= 1 TO np
12080  tabla$(i,2)=STR$(0)
12090 NEXT i
12100 RETURN
14000 REM Listado de Clientes *************************
14010 PRINT CLS$
14020 PRINT FN LOCATE$(10,5);:PRINT"1. Listado por pantalla"
14030 PRINT FN LOCATE$ (15,5);:PRINT"2. Listado por impresora"
14040 GOSUB 54000:'pulsar tecla
14050 a=VAL(a$)
14055 PRINT CLS$
14060 IF a=1 THEN GOSUB 56000:'pantalla
14070 IF a=2 THEN GOSUB 57000:'impresora
14080 RETURN
```

```
16000 REM FIN ********************************************
16010 OPEN "O",1,"ficlient"
16020 PRINT #1,np
16030 FOR i=1 TO np
16040   FOR j=1 TO 2
16050   PRINT #1, tabla$(i,j)
16060 NEXT j
16070 NEXT i
16080 CLOSE 1
16090 PRINT CLS$:END
50000 REM lectura ****************************************
50010 PRINT CLS$
50020 REM lee el fichero si se introduce
50030 REM el disco que contiene el fichero llamado
50040 REM ficlient
50050 OPEN "I",1, "ficlient"
50060 INPUT #1,np
50070 FOR i=1 TO np
50080 FOR j=1 TO 2
50090 LINE INPUT #1,tabla$(i,j)
50100 NEXT j
50110 NEXT i
50120 CLOSE 1
50130 PRINT CLS$
50140 RETURN
51000 REM confirmacion ***********************************
51010 PRINT FN LOCATE$ (17,50);:PRINT"Confirmacion (S/N)";
51020 a$=INKEY$
51030 IF a$="" THEN 51020
51040 a$=UPPER$(a$)
51050 IF a$="S"  THEN con$="si"
51060 IF a$="N" THEN con$="no"
51070 IF a$<>"S" AND a$<>"N" THEN 51020
51080 RETURN
52000 REM localizacion ************************************
52010 PRINT FN LOCATE$ (10,5);:INPUT "Introducir nombre";n$
52020 n$=UPPER$(n$)
52030 FOR i=1 TO np
52040 IF n$=UPPER$(tabla$(i,1)) THEN 52060
52050 NEXT i
52060 IF i=np+1 THEN enc$="no" ELSE enc$="si"
52070 RETURN
53000 REM retardo *****************************************
53010 FOR t=1 TO 2000
53020   NEXT t
53030 RETURN
54000 REM pulsar tecla ************************************
```

```
54010 PRINT FN LOCATE$ (25,35);:PRINT"pulse una tecla ";
54020 a$=INKEY$:IF a$="" THEN 54020
54030 RETURN
55000 REM titulo**********************************
55010 PRINT CLS$
55020 PRINT FN LOCATE$ (1,43):PRINT op$
55030 RETURN
56000 REM Pantalla ***************************************
56010 FOR i=1 TO np
56020  PRINT UPPER$(tabla$(i,1)) ;STRING$(40-LEN(TABLA$(I,1)),"."
;TAB(45);USING "#######.##";VAL(TABLA$(I,2))
56030 NEXT i
56040 PRINT:PRINT"Pulse una tecla"
56050 IF INKEY$="" THEN 56050
56060 RETURN
57000 REM Impresora ***************************************
57010 LPRINT "NOMBRE";TAB(45);"***VENTAS***"
57020 LPRINT
57030 FOR i=1 TO np
57040 LPRINT UPPER$(tabla$(i,1));STRING$(40-LEN(TABLA$(I,1)),".")
TAB(45);USING "#######.##";VAL(TABLA$(I,2))
57050 NEXT i
57060 RETURN
58000 REM Nombre inexistente-pitido
58010 PRINT "Nombre inexistente"
58020 PRINT CHR$(7)
58030 RETURN
```

14. FICHEROS ALEATORIOS

La forma de trabajo habitual en ficheros secuenciales consiste en cargar los datos almacenados en disco en una tabla y posteriormente trabajar con ella, realizando inserciones, modificaciones, eliminación de registros, etc. La última operación consiste en grabar la información que contiene la tabla en memoria. Los datos quedan grabados secuencialmente en memoria.

Cuando el número de datos a grabar es alto, la memoria central del ordenador se ve incapaz de almacenar el programa y la tabla. Por otra parte, el acceso a un determinado registro, se debe realizar leyendo todos los anteriores con lo que el tiempo en la búsqueda puede ser excesivamente elevado. Es en estos casos cuando se hace conveniente la utilización de ficheros aleatorios.

14.1. Apertura de un fichero.

Para abrir un fichero se utiliza la instrucción OPEN con el siguiente formato:

OPEN "R",<n>,"<nombre de fichero>",<lonreg>

donde los argumentos tienen el siguiente significado:

R: Indicativo de fichero aleatorio (Random).

n: Número de fichero. Tiene que estar comprendido entre uno y tres.

nombre de fichero: Es el nombre con el que aparecerá el fichero en el disco.

lonreg: Este parámetro es opcional. Indica la longitud que tiene cada registro en caracteres. Por defecto toma el valor 128. Si se quiere utilizar un número más alto es necesario utilizar la instrucción MEMORY que se estudiará en este capítulo.

Cuando se abre un fichero aleatorio se crea si no hay ninguno con el mismo nombre grabado en disco. De haber algún fichero con el mismo nombre, se deja preparado para ser utilizado.

14.2. Estructura de un fichero aleatorio.

Un fichero aleatorio está compuesto de registros de longitud fija según se indica en el siguiente esquema:

Campo 1	Campo 2	Campo 3	Campo 4	Campo 5	Campo 6
Campo 1	Campo 2	Campo 3	Campo 4	Campo 5	Campo 6
Campo 1	Campo 2	Campo 3	Campo 4	Campo 5	Campo 6
Campo 1	Campo 2	Campo 3	Campo 4	Campo 5	Campo 6
Campo 1	Campo 2	Campo 3	Campo 4	Campo 5	Campo 6
Campo 1	Campo 2	Campo 3	Campo 4	Campo 5	Campo 6

...

Campo 1	Campo 2	Campo 3	Campo 4	Campo 5	Campo 6

La longitud total de cada registro (la suma de los campos) es siempre la misma para todos los registros. Si bien puede haber registros con distinta estructura en sus campos.

Para definir la estructura de un registro se utiliza la instrucción FIELD con el siguiente formato:

```
FIELD <numfich>,n1 AS c1$,n2 AS c2$,n3 AS c3$
```

donde:

numfich: Número de fichero que va a contener la estructura de registro que se define.

ni: Número de caracteres que componen el campo i.

c1$, c2$, c3$: Nombre de los campos que intervienen en el registro. (Todos los campos deben ser alfanuméricos). Estas variables son llamadas *variables de campo*.

Ejemplo.

Vamos a diseñar un fichero aleatorio en el que cada registro tenga el siguiente formato:

nº identificativo: 5 caracteres.

nombre: 30 caracteres.

domicilio: 40 caracteres.

dni: 9 caracteres.

población: 15 caracteres.

La longitud del registro será por tanto:

5+30+40+9+15=99

Entonces escribiremos para la apertura del fichero:

```
OPEN "R",1,"personal",99
```

Una vez abierto el fichero, definimos la estructura de sus registros de la forma siguiente:

```
FIELD 1,5 AS ni$,30 AS nom$,40 AS domi$,9 AS dni$,15 AS pob$
```

14.3. Escritura en un fichero aleatorio.

Para grabar un registro en un fichero aleatorio hay que asignar valores a las variables de campo y después grabar el registro.

Para dar contenidos a las variables de campo se utilizan las siguientes instrucciones:

*LSET : Da un valor a una variable de campo truncando por la derecha, es decir, que si intentamos introducir en un campo un número mayor de caracteres de la longitud que se ha definido para el campo, se pierden los últimos caracteres.

Formato:

LSET campo$=<expresión alfanumérica>

*RSET :Actúa de la misma manera que LSET pero truncando por la izquierda.

Formato:

RSET campo$=<expresión alfanumérica>

*MID$:Toma caracteres intermedios de una expresión alfanumérica:

Formato:

MID$ (<campo$>,<primero>,<longitud>)

Extrae el nº de caracteres indicados por la variable longitud a partir del indicado por la variable primero.

Lo habitual es truncar por la derecha utilizando la instrucción LSET.

En el ejemplo anterior escribiríamos las siguientes instrucciones para rellenar las variables de campo.

```
INPUT "Introducir numero de persona";np
INPUT "Introducir nombre";nombre$
INPUT "Introducir domicilio";domicilio$
INPUT "Introducir numero de dni";dni
INPUT "Introducir poblacion";poblaci$
LSET ni$=str$(np)
LSET nom$=nombre$
LSET domi$=domicilio$
LSET dni$=str$(dni)
LSET pob$=poblaci$
```

Obsérvese que los contenidos de las variables numéricas np y dni han de convertirse a tipo alfanumérico antes de asignarlos a una variable de campo. En este capítulo se estudiará un método más eficiente de tratamiento de las variables numéricas.

Una vez que se han introducido valores en las variables de campo se debe proceder a su grabación, operación que se realiza con la instrucción PUT que tiene el siguiente formato:

PUT <numfil>,<numreg>

Donde numfil indica el nº de fichero en que se graba el registro y numreg indica la posición en que se encontrará el registro en el fichero. Este segundo argumento es opcional. Por defecto, se graba el registro después del último registro grabado o leído.

14.4. Cierre de un fichero aleatorio.

Para cerrar un fichero de acceso aleatorio se escribe la instrucción CLOSE. Esta instrucción sin parámetros hace que se cierren todos los ficheros. Para cerrar un fichero determinado se escribe después de la instrucción CLOSE el número del fichero que se desea cerrar.

14.5. Lectura de los registros de un fichero.

Los datos grabados en el fichero se leen introduciendo los datos de un registro determinado en las variables de campo. Esta operación se realiza con la instrucción GET. El fichero debe estar abierto previamente según el método estudiado. Asimismo, debe estar definida la estructura del registro mediante una instrucción FIELD que es conveniente que sea la misma que se utilizó cuando se creó el registro.

La instrucción GET puede tener dos formatos distintos:

#GET <nº de fichero>

Se toma el registro siguiente al último escrito con PUT o leído con GET en el fichero cuyo número se especifica.

*GET <nº de fichero>,<nº de registro>

Se lee el registro que ocupa la posición indicada por el número de registro del fichero señalado por el número de fichero.

14.6. Tratamiento de variables numéricas.

Las variables numéricas pueden ser almacenadas de una forma más eficiente que la estudiada. Existen en BASIC varias funciones de conversión de números a cadena literal y viceversa que permiten almacenar con menos bytes una cantidad numérica que si se almacena como tipo alfanumérico.

** La función MKS$ permite convertir un número en una cadena que ocupe cuatro caracteres. De esta manera, el número de caracteres que se deben reservar para una variable de campo numérica (instrucción FIELD) es de cuatro caracteres.

Su formato es el siguiente:

<variable de campo>=MKS$(<variable numérica>)

Si por ejemplo se quisiera almacenar un nº representante del documento de identidad de una persona se escribiría:

```
INPUT "Introducir D.N.I.";doc
dni$=MKS$(doc)
```

La variable dni$ sería la variable de campo y deberían reservarse para ella cuatro caracteres en la instrucción FIELD.

** La función MKI$ funciona igual que MKS$ con la particularidad de que el número que se convierte a tipo alfanumérico ha de ser entero. Los caracteres que se deben dejar en la variable de campo en este caso es de dos.

Cuando se han usado estas funciones para almacenar datos en el fichero, es preciso usar las funciones CVS y CVI - que

convierten la información numérica almacenada en numérica - para recuperar los datos. La forma de operar es la siguiente:

1. Con la instrucción GET se obtiene un registro. (Y por tanto quedan rellenas las variables de campo.)

2. Cada variable que fuese almacenada utilizando las funciones MKS$ y MKI$ se recupera con las funciones CVS y CVI respectivamente.

Los formatos de las funciones CVS y CVI son los siguientes:

<variable numérica>=CVS (variable de campo)

<variable numérica>=CVI (variable de campo)

Siguiendo con el ejemplo del almacenamiento del nº del DNI podríamos escribir la siguiente secuencia de instrucciones para recuperarlo y poder trabajar con él en forma numérica:

```
INPUT "Introducir nº de persona";np
GET 1,np
dni=CVS(dni$)
```

En la variable ***dni*** ya se encuentra el dato en forma numérica. Hay que hacer constar que si se hubiera utilizado la función MKI$ habría que haber escrito CVI en lugar de CVS.

14.7. Instrucción MEMORY.

La instrucción Memory sirve para cambiar el máximo número de ficheros y para aumentar la máxima longitud standard de registros (128 caracteres).

Formato

MEMORY,<dir.alta>,<tamaño de pila>,<numero de ficheros>,<longitud de registro>

Todos los argumentos son opcionales. Los dos primeros son para modificar atributos de la memoria. El primero, para cambiar su dirección más alta para BASIC y el segundo indica número de bytes que se reservan para la pila.

El tercer parámetro especifica el máximo número de ficheros utilizables al mismo tiempo. (Sólo es necesario este parámetro cuando el número es mayor de tres).

El cuarto parámetro indica la máxima longitud de un registro para un fichero de acceso aleatorio. (Hay que escribir este parámetro cuando la longitud es superior a 128 caracteres).

Ejemplo 1

Utilizar la instrucción MEMORY para que:

1. Se puedan tener abiertos 5 ficheros simultáneamente.

2. La longitud de cada registro de un fichero aleatorio pueda ser de 150 caracteres.

3. Se puedan utilizar cinco ficheros a la vez y que la longitud de cada registro de un fichero aleatorio pueda ser de 150 caracteres.

*** 1.

```
MEMORY,,,5
```

*** 2.

```
MEMORY,,,,150
```

*** 3.

```
MEMORY,,,5,150
```

Realizar una agenda con opciones de altas, bajas, consultas y modificaciones utilizando un fichero aleatorio.

Para cada persona se necesita almacenar su nº identificativo, nombre, domicilio y teléfono.

```
-----------------------------------------------
| Ident |  Nombre |  Domicilio  |   Telefono  |
-----------------------------------------------
    4        30          30              10
```

```
10 REM Agenda ***Fichero aleatorio
20 DEF FN locate$(y,x)=CHR$(27)+"Y"+CHR$(32+y)+CHR$(32+x)
30 cls$=CHR$(27)+"E"+CHR$(27)+"H"
40 PRINT cls$
50  IF  FIND$("datos")=""  THEN  GOSUB  780  ELSE  OPEN
"R",1,"datos",74
60 FIELD 1,4 AS num$,30 AS nom$,30 AS domi$,10 AS tel$
70 FIELD 1,74 AS emple$
80 REM menu
90 PRINT cls$
100 PRINT FN locate$(12,10);"1. Altas"
110 PRINT FN locate$(14,10);"2. Bajas"
120 PRINT FN locate$(16,10);"3. Consultas"
130 PRINT FN locate$(18,10);"4. Impresion"
140 PRINT FN locate$ (20,10);"5. Fin";
150 PRINT FN locate$ (30,30);"Pulsar opcion";
160 a$=INKEY$:IF a$="" THEN 160
170 a=VAL(a$)
180 ON a GOSUB 200,410,510,570,640
190 GOTO 80
200 REM Altas
210 PRINT cls$
220 INPUT "Introducir numero de empleado";n
230 PRINT:PRINT
240 INPUT "Introducir nombre de empleado";ne$
250 PRINT:PRINT
260 INPUT "introducir domicilio ";d$
270 PRINT:PRINT
280 INPUT "Introducir telefono";t$
290 PRINT:PRINT:PRINT
300 PRINT"Confirmacion (S/N)";
310 a$=INKEY$:IF a$="" THEN 310
320 IF a$="S" OR a$="s" THEN 350
330 IF a$="N" OR a$="n" THEN RETURN
340 GOTO 310
```

```
350 LSET num$=MKS$(n)
360 LSET nom$=ne$
370 LSET tel$=t$
380 LSET domi$=d$
390 PUT 1,n
400 RETURN
410 REM Bajas
420 PRINT cls$
430 INPUT "Introducir numero de empleado";n
440 IF n<1 OR n>100 THEN 680
450 PRINT"Confirmacion(S/N)?";
460 a$=INKEY$:IF a$="" THEN 460
470 IF a$="S" OR a$="s" THEN GOTO 480 ELSE IF a$="N" OR
a$="n" THEN RETURN ELSE 460
480 LSET emple$=""
490 PUT 1,n
500 RETURN
510 REM consulta
520 PRINT cls$
530 GOSUB 680
540 PRINT cls$
550 GOSUB 720
560 RETURN
570 PRINT cls$:GOSUB 680
580 IF nom$=STRING$(30," ") THEN PRINT :PRINT:PRINT"Persona
inexistente":GOSUB 860:RETURN
590 LPRINT "Nombre :";nom$
600 LPRINT:LPRINT "direccion :";domi$
610 LPRINT:LPRINT "telefono :";tel$
620 LPRINT:LPRINT "Nº empleado :";CVS(num$)
630 RETURN
640 REM fin
650 CLOSE
660 PRINT cls$
670 END
680 INPUT "Introducir numero de empleado";n
690 IF n<1 OR n>100 THEN 680
700 GET 1,n
710 RETURN
720 PRINT FN locate$(12,10);"Nombre:";TAB(30);UPPER$(nom$)
730 PRINT FN locate$(14,10);"Domicilio";TAB(30);UPPER$(domi$)
740 PRINT FN locate$(16,10);"Telefono";TAB(30);tel$
750       PRINT       FN       locate$(18,10);"Numero       de
empleado";TAB(30);ROUND(CVS(num$))
760 GOSUB 860:REM esperar pulsacion
770 RETURN
780 REM creacion fichero vacio
```

```
790 OPEN "R",1,"datos"
800 FIELD 1,74 AS perso$
810 LSET perso$=""
820 FOR i=1 TO 100
830 PUT 1
840 NEXT i
850 RETURN
860 PRINT:PRINT:PRINT "Pulse una tecla";
870 WHILE INKEY$="":WEND
880 RETURN
```

14.8. Tratamiento Secuencial de un fichero aleatorio.

En el ejemplo anterior la búsqueda de un registro se realiza siempre por un número identificativo del registro (el número de empleado) con lo que el acceso es directo. También sería posible localizar un registro a partir de cualquier otro campo, por ejemplo por el nombre del empleado. El método sería similar al utilizado en ficheros secuenciales. Esto es, se irían obteniendo los registros del fichero, comenzando por el primero y en el momento en que coincidiese el nombre buscado con el campo del registro donde se encuentra el nombre (nom$), se detendría el proceso.

Ejemplo

```
1000 REM Busqueda por nombre
1010 INPUT "Introducir nombre del empleado";n$
1020 IF LEN (n$)<30 THEN n$=n$+" ":GOTO 1020
1030 IF LEN (n$)>30 THEN n$=LEFT$(n$,30)
1040 FOR i=1 TO 100
1050   GET 1,i
1060   IF nom$=n$ THEN GOTO 1080
1070 NEXT i
1080 REM Tratamiento del registro i
```

De esta manera se localiza un determinado registro a partir de cualquier campo. Mediante la instrucción 1020 se rellenan de blancos los caracteres que falten hasta llegar a 30 (longitud de la variable de campo nom$) en el nombre que se introduzca para poder realizar posteriormente la comparación con la variable de campo.

El presente libro ha sido desarrollado, en base a la amplia experiencia pedagógica que

SPEN, S.A.

ha acumulado a lo largo de tres años de impartir clases de Informática en más de cien centros educativos y a miles de alumnos de edades comprendidas entre los ocho y veintitres años

SPEN,SA

Infanta Mª Teresa, 23
MADRID

TASOFT