Unidad 6
ESTRUCTURAS EXTERNAS: Ficheros
Desarrollo de la unidad : 36 h

Prácticas :
Ejercicios :
Conceptos:
Memoria principal, Memoria Secundaria, Fichero, Sistema de ficheros, Registros, tipos de ficheros, operaciones con ficheros, organización secuencial, encadenada, indexada
Ficheros planos, texto y estructurados
Acceso a Bases de datos

INTRODUCCION

Conceptos Previos

- Memoria principal (Limitada, muy rápida, volátil)
- Memoria secundaria (Menos limitada, más lenta, no volátil)

Un programa no puede acceder directamente a la memoria secundaria, tiene que transferir la información de la memoria secundaria a la memoria principal, para que el programa pueda modificar los datos almacenados.

Diferencias entre ficheros y Tablas

	Tablas	**Fichero**
Capacidad	Limitada (Memoria RAM)	Menos Limitada (Memoria Secundaria Disco, Cinta, etc)
Permanencia	No permanente	Permanente
Acceso	Inmediato	A través de Sistema Operativo
Tamaño	Fijo	Variable
Dimensiones	N dimensiones	Una dimensión

Fichero: Es un estructura informática para el almacenamiento de datos en memoria secundaria gestionado por el sistema operativo. Un fichero es un conjunto de sectores físicos de un dispositivo de memoria secundaria que guardan información sobre un tema concreto. El S.O. es el encargado de gestionar que sectores pertenece a cada fichero, si están ocupado o libres, si tienen fallos, si el sector esta completo, cual es el siguiente sector de un fichero, etc.

La estructura de control que gestiona la información sobre ficheros, sectores, ocupación, etc, se denomina Sistema de ficheros. Existen distintos sistemas de ficheros según el sistema operativo con el que trabajemos: FAT16, FAT32, NTFS, EXT2, RAISER, NFS, SySV, OS2... Existen también sistemas de ficheros estándar, independientes del sistema operativo, como el ISO 9660 que corresponde con el formato habitual de los CD-ROM de datos. Para poder trabajar con una unidad de almacenamiento, debemos crear la información sobre sistema de ficheros que reconoce el sistema operativo, a este proceso se le denomina comúnmente formatear.

Un fichero puede contener cualquier tipo de información: instrucciones en C (c, cpp, h), en código máquina (.exe, com, lib, obj, dll) , imagenes en distintos formatos (jpg, gif, bmp), documentos de texto sin formato (txt), con un formato especifico (rtf, doc, html), sonido (wav, mp3), video (avi,divx), fichas de los clientes, una lista de alumnos, el registro de windows, etc.

La gran capacidad de la memoria RAM que ofrecen actualmente los ordenadores hace que gran parte del tratamiento que antes se realizaba en base a ficheros, ahora se realice con estructuras en memoria RAM, grabando la información a fichero sólo cuando es necesario. Para trabajar con grandes volúmenes de datos se utiliza, por el contrario, los llamados Sistemas de Gestión de bases de Datos. Los SGBD facilitan enormemente la labor al programador, ofreciendo un buen rendimiento y velocidad en acceso a los datos.

¿Que debe controlar el sistema operativo de una unidad de almacenamiento?

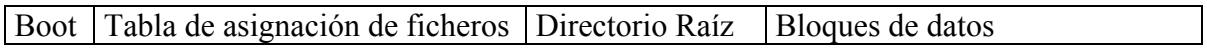
Hay que tener control de los bloques libres y ocupados por cada fichero, mantener la información sobre los directorios, manejar los buffers en las operaciones de entrada y salida, controlar que ficheros están abiertos, por que programas, almacenar los premisos, permitir enlaces, controlar el acceso compartido a un mismo fichero....

Una unidad de almacenamiento (Disco duro) puede tener varias particiones, cada partición tiene su propio sistema de ficheros independiente. Un sistema operativo puede controlar distintas particiones con sistemas de ficheros diferentes. Ej.- GNU/Linux (Ext2, ReiserFS, FAT) o Windows 2000 Server (FAT, NTFS). Generalmente es más seguro tener varias particiones que una sola muy grande.

El sistema de ficheros FAT

Es el tipo de partición común de MS-DOS y Windows 98. Es un sistema de ficheros bastante rápido y seguro aunque no permite el control de usuarios en el acceso a los ficheros, por eso en sistemas servidores (NT o W2K) lo habitual es utilizar particiones NTFS.

Esquema del una partición FAT:



En las particiones FAT la información de control de los archivos se almacena en los directorios. La asignación de espacios se realiza mediante la gestión de una lista encadenada de bloques en la llamada tabla de asignación de ficheros. El directorio raíz es un directorio especial que se tiene un tamaño fijo.

Información de un directorio en un partición FAT (Un archivo especial)

- Nombre del archivo
- Extensión
- Atributos (Sólo lectura, oculta, sistema, normal)
- Tamaño
- Hora y fecha de creación / modificación / acceso
- Puntero al primer bloque ocupado por el fichero en la Tabla de asignación de Ficheros

Un fichero no siempre se guarda en posiciones consecutivas, sino que puede utilizar varios bloques en distintas posiciones. El recorrido completo del archivo puede implicar el posicionamiento en distintas posiciones del disco duro. Este fenómeno se conoce como fragmentación y repercute negativamente en el rendimiento de las operaciones de entrada y salida.

TIPOS DE ARCHIVOS

Permanentes o Maestros

Son ficheros que se mantienen a la largo de todo el tiempo (meses y años, mientras se utilice la aplicación o programa que los trata). Pueden sufrir modificaciones aunque siempre serán parciales. A su vez, los podemos subdividir según la frecuencia de actualización en:

- ***Ficheros de constantes***. Pensemos por ejemplo en un fichero que contenga una tabla de logaritmos, una tabla periódica, códigos postales, etc., su contenido permanece siempre inalterable.
- ***Ficheros de situación***. Son ficheros que se modifican con más frecuencia y que contienen la situación actualizada. Guardan información susceptible a sufrir frecuentes modificaciones o alteraciones. Ej.- Clientes de la empresa, saldos bancarios, etc
- ***Ficheros históricos***. Contienen información referente a resultados de operaciones, información resumida, etc., a partir de las cuales se obtiene y confecciona un resumen o resultado final. A partir de ellos pueden elaborarse estadísticas. Ej.- Apuntes contables, movimientos de almacén, ventas mensuales, etc.

Temporales

-De movimientos o transacciones **(transaction file).**

Son ficheros de corta vida, que contienen información a cerca de los movimientos y transacciones. Suelen servir para la consulta de los ficheros permanentes y producir junto a ellos la información final. Ej.- Lista de entradas en almacén, pedidos, etc.

-De maniobra o trabajo **(work file).**

La mayoría de ellos se utilizar para realizar una operación determinada y luego desaparecen. Contienen información referente a resultados intermedios que a su vez son suministrados a otros programas. Ej.- Un fichero auxiliar para reordenar un fichero maestro de clientes.

OPERACIONES SOBRE FICHEROS

Un programa no puede trabajar directamente con la información contenida en un fichero. Las operaciones sobre ficheros se hacen en base a llamadas a funciones del sistema operativo, que es quien controla directamente el acceso al sistema de ficheros y al hardware de almacenamiento secundario.

Operaciones básicas (Implementadas directamente por el Sistema Operativo)

- Apertura - open
- Cierre - close
- Leer - read
- Escribir - write
- Situarse - lseek
- Creación - creat
- Borrado – unlink, remove
- Truncado – trunc (Borrar los últimos bytes del fichero)
- Fijar permisos - chmod

OJO*: No se puede insertar o recortar directamente la información que se guarda en un fichero. El único método de cambiar el tamaño de un fichero es quitar o poner bytes al final del mismo.*

Las operaciones sobre memoria secundaria son miles de veces más lentas que sobre memoria RAM. Para mejorar el rendimiento de las operaciones de E/S sobre ficheros, el sistema operativo gestiona un conjunto de almacenamientos intermedios o buffers que permiten que muchas operaciones de E/S se realicen en memoria RAM, en vez de tener que acceder constantemente a la memoria Secundaria. Esta caracteriza puede provocar que después de un apagado repentino del ordenador, se hayan perdido datos, pues aunque nuestro programa trabaje sólo con ficheros, puede que las operaciones se hayan hecho sobre el buffer y que todavía el sistema operativo no haya volcado a disco. Hasta que no cerramos el fichero no tenemos la seguridad que todas la información escrita se ha guardado en memoria secundaria.

Operaciones Secundarias (Mediante un conjunto de operaciones básicas)

- Copiado o Duplicación
- Concatenación
- Ordenación
- Intersección
- Mezcla
- Partición o rotura
- Compactación o Empaquetado

Registros y Operaciones sobre registros.

Gran parte de los ficheros de datos contienen información organizada en una serie de registros. Un registros es una serie de datos de distinto tipo (Ej.- struct del lenguaje C).

Elementos de un registro: (campo, subcampo, campo clave)

Cuando un fichero está formado por una serie de registros se suelen definir un conjunto de operaciones especificas que afectan a dichos registros.

- Consulta (Lectura de un registro para acceder a su contenido)
- Modificación (Modificar el contenido del registro y escribirlo en el fichero)
- Alta (Escribir un nuevo registro en el fichero: implica una escritura al final o una reorganización)
- Baja (Implica escribir en el fichero, marcarlo mediante un borrado lógico, posteriormente se suele reorganizar el fichero y truncar el espacio libre)

- Normalmente las altas y bajas son las operaciones más costosas en tiempo de ejecución.

Tipos de registros

- Tamaño y estructura fija. (Ídem que *struct* de C, es el tipo más común)

- Tamaño fijo y estructura variable (Ídem que la *union* de C)

- Tamaño variable y estructura fija (Normalmente el último campo es de tamaño variable, ejemplo la mayoría de los formatos de imagen.

- Tamaño y estructura variable. (Suele existir una serie de campos fijos que define la estructura del resto del registro)

FICHEROS EN LENGUAJE C.

El lenguaje C, al no ser un lenguaje pensado para desarrollar software de gestión, ofrece una manejo muy básico de los ficheros, apenas algo más que lo que implementa directamente el sistemas operativo. Sin embargo, es posible realizar un tratamiento más complejo sirviéndose de librerías especificas o mediante el acceso a bases de datos

Tipos de archivos :

- **Texto:** Tiene una estructura muy sencilla: caracteres ASCII separados por salto de línea. Leemos y escribimos letras, líneas, cadenas de caracteres, etc. Generalmente se realiza un acceso secuencial.

- **Binarios:**
  Leemos y escribimos un número de bytes concretos. Acceso secuencial o directo.

  - Planos : Cualquier fichero, sin considerar la estructura de la información que contiene. Sólo es una lista consecutiva de bytes / octetos .

  - Estructurados: Con una estructura más menos compleja. Normalmente una serie de registros de un mismo tipo. Ej.- Un fichero de clientes, formado por un una serie de registro con la información de cada uno de los clientes de nuestra empresa.

Funciones comunes:

- fopen – Abrir el fichero
- fclose – Cerrar el fichero
- fflush – Vaciar el buffer de E/S
- rewind – Volver al principio del fichero.

FICHEROS DE TEXTO

Generalmente siempre se realiza un tratamiento secuencial, según el siguiente esquema básico:

```
Abrir
Mientras (Condición)
      Leer o escribir
Fin-mientras
Cerrar
```

Ficheros de texto predefinidos:

stdin -> Corresponde con la entrada estándar (Normalmente teclado)
stdout-> Corresponde con la salida estándar (Normalmente la pantalla)
stderr-> Corresponde con la salida estándar de errores (Normalmente la pantalla)

Funciones básicas:

fgets, Leer una cadena
fputs, Escribir una cadena
fgetc, Leer un carácter.
fputc, Escribir un carácter
feof, Detectar fin de fichero
fscanf, Leer según formato.
fprintf, Escribir según formato.

Ejercicios (Consultar página web):

1. Mostrar el contenido de un fichero de texto
2. Grabar o Añadir información a un fichero de texto
3. Contar los caracteres, líneas y palabras de un fichero de texto
4. Concatenar dos ficheros
5. Mostar parte de un fichero
6. Copiar selectivamente
7. Mostrar las primeras o las últimas líneas de un fichero
8. Ordenar un fichero de texto mediante una tabla.

FICHEROS BINARIOS

Se pueden realizarse acceso directo o acceso secuencial, pueden guardar cualquier tipo de información en cualquier formato.

Funciones básicas:
fread, - Leer una cantidad de datos concretos
fwrite – Escribir una cantidad de datos concretos
fseek – Situarse en una posición determinada del fichero
ftell – Obtener la posición actual donde estamos situados.

Ejercicios propuestos:

Fichero planos

1. Copiar dos ficheros cualquiera
2. Partir un fichero en partes

Ficheros de registros

1- Mostar el contenido de un ficheros de registros
2- Grabar en un fichero de registros.
3- Informe de un fichero
4- Ordenar un fichero directo mediante una tabla
5- Realizar un mantenimiento sencillo
6- Realizar un mantenimiento complejo
7- Tratar un fichero encadenado.
8- Procesar algún fichero multimedia.

PASOS PARA OPERAR CON FICHEROS

Para trabajar con ficheros en C podemos utilizar directamente las llamadas a sistema utilizando en los llamados descriptores de ficheros o bien utilizar la librería *stdio*. Esta librería, basada en la anteriores llamada, ofrecen un amplio conjunto de funciones que facilitan la labor del programador. La estructura de datos que gestiona los ficheros (FILE *) se denomina STREAM o flujos de datos.

Los pasos habituales para realizar operaciones con un fichero son los siguientes:

1) Definir una variable tipo fichero. Esto se hace en C declarando un puntero a FILE. La variable puntero apunta a una estructura llamada FILE. Esta estructura está definida en el fichero stdio.h y contiene toda la información necesaria para poder trabajar con un fichero. El contenido de esta estructura es dependiente de la implementación de C y del sistema, y en general no es necesario que el programador la conozca.
Ejemplo:

```
FILE *pf; /* pf es un puntero de fichero */
```

2) Abrir el fichero. Esto se hace con la función fopen() cuyo prototipo se encuentra en el fichero stdio.h. Su sintaxis es la siguiente:

```
FILE *fopen (char *nombre_fichero, char *modo);
```

Si el fichero con nombre nombre_fichero no se puede abrir, devuelve el valor NULL.

El parámetro nombre_fichero puede contener la ruta completa del fichero pero teniendo en cuenta que la barra invertida (\)hay que repetirla en una cadena de caracteres. Si trabajamos en entorno UNIX, se utiliza la otra barra (/).

```
Pf = fopen ("C:\\Miarchivos\\datos.dat","r");
```

Los valores válidos para el parámetro modo son:

Modo	Interpretación
r	Abrir un fichero texto para lectura
w	Abrir/Crear un fichero texto para escritura
a	Añadir a un fichero texto
rb	Abrir un fichero binario para lectura
wb	Abrir/Crear un fichero binario para escritura
ab	Añadir a un fichero binario
r+	Abrir un fichero texto para lectura/escritura
w+	Crear un fichero texto para lectura/escritura
a+	Abrir un fichero texto para lectura/escritura
rb+	Abrir un fichero binario para lectura/escritura
wb+	Crear un fichero binario para lectura/escritura
ab+	Abrir un fichero binario para lectura/escritura

Si se utiliza fopen() para abrir un fichero para escritura, entonces cualquier fichero que exista con ese nombre es borrado y se comienza con un fichero nuevo. Si no existe un fichero con ese nombre, entonces se crea uno. Si lo que se quiere es añadir al final del fichero, se debe utilizar el modo a. Si no existe el fichero, devuelve error. Si abrimos un fichero para operaciones de lectura es necesario que el fichero exista previamente. Si no existe devuelve error. Finalmente, si se abre un fichero para operaciones de lectura/escritura, no se borra en caso de existir. Sin embargo, si no existe se crea.

Ejemplo:

```
FILE *pf;
pf = fopen (c:\\autoexec.bat,"r");
if (pf == NULL) /* siempre se debe hacer esta comprobación */
  {
    puts (No se puede abrir fichero.);
    exit (1);
  }
```

3) Realizar las operaciones deseadas con el fichero, normalmente escritura y lectura. Existen distintas funciones según trabajemos con ficheros de texto o binarios.
El procesamiento de los datos de un fichero se hace casi siempre mediante ciclos MIENTRAS pues generalmente no se conoce previamente el tamaño o número de datos que contiene un fichero. Después de cada operación de lectura hay que comprobar si se ha realizado con éxito.

4) Cerrar el fichero. Aunque el sistema operativo, cierra automáticamente todos los ficheros abiertos al terminar el programa, es aconsejable cerrarlos explícitamente. Esto se hace con la función fclose() cuyo prototipo es:

```
int fclose (FILE *pf);
```

La función fclose() cierra el fichero asociado con el flujo pf y vuelca su buffer.

Si fclose() se ejecuta correctamente devuelve el valor 0. La comprobación del valor devuelto no se hace muchas veces porque no suele fallar.

Ejemplo:

```
FILE *pf;
if ((pf = fopen ("prueba", "rb")) == NULL)
  {
    puts ("Error al intentar abrir el fichero.");
    exit (1);
  }
/* ... */
if (fclose (pf) != 0)
  {
    puts (Error al intentar cerrar el fichero.);
    exit (1);
  }
```

FUNCIONES BÁSICAS DE MANIPULACIÓN DE FICHEROS

feof Indica si se ha alcanzado el final de fichero.

Sintaxis:

```
int feof (FILE *flujo);
```

Una vez alcanzado el final del fichero, las operaciones posteriores de lectura devuelven EOF hasta que se cambie la posición del puntero del fichero con funciones como rewind() y fseek().

Ejemplo:

```
  /* supone que pf se ha abierto como fichero
     binario para operaciones de lectura */

letra = getc (pf);  // Leo una letra
while (! feof (pf)) // Mientras no haya llegado al final de fichero
    {
    putchar(letra); // muestro la letra leida
    letra = getc (pf); // Leo la siguiente letra
    }
```

fflush Vuelca el buffer de un fichero.

Sintaxis:

```
int fflush (FILE *flujo);
```

Si el flujo está asociado a un fichero para escritura, una llamada a fflush() da lugar a que el contenido del buffer de salida se escriba en el fichero. Si flujo apunta a un fichero de entrada, entonces el contenido del buffer de entrada se vacía. En ambos casos el fichero permanece abierto.

Devuelve EOF si se detecta algún error.

Ejemplo:

```
  /* supone que pf está  asociado
     con un fichero de salida */

  fprintf(pf," El valor es %d", num);
  fflush (pf);
```

fgetc, getc Obtiene un carácter del fichero.

Sintaxis:

```
int fgetc (FILE *flujo);
```

La función fgetc() devuelve el siguiente carácter desde el flujo de entrada e incrementa el indicador de posición del fichero. El carácter se lee como un unsigned char que se convierte a entero.

Si se alcanza el final del fichero, fgetc() devuelve EOF. Recuerda que EOF es un valor entero. Por tanto, cuando trabajes con ficheros binarios debes utilizar feof() para comprobar el final del fichero. Si fgetc() encuentra un error, devuelve EOF también.

Ejemplo:

```
  /* supone que pf está asociado
     con un fichero de entrada */

  letra = fgetc(pf);
  while ((letra!= EOF)
    {
    printf (%c, letra);
    letra = fgetc(pf);
    }
```

NOTA: getchar() es una macro de la función fgetc(stdin);

fgets Lee una cadena de caracteres del fichero

Sintaxis:

```
  char *fgets (char *s, int n, FILE *flujo);
```

La función fgets() lee hasta n-1 caracteres desde el flujo y los sitúa en la cadena apuntada por s. Los caracteres se leen hasta que se recibe un carácter de salto de línea o un EOF o hasta que se llega al límite especificado. Después de leídos los caracteres, se sitúa en la cadena el carácter '\0' inmediatamente después del último carácter leído. *A diferencia la función gets, se guarda el carácter de salto de línea, siendo el último carácter de la cadena s.*

Si fgets() tiene éxito devuelve la dirección de s; se devuelve un puntero nulo (NULL) cuando se produce un error.

Ejemplo:

```
  /* supone que pf está  asociado con
     un fichero de entrada */

  char msg[100]

  // Lee cadenas hasta que la función falla
  while ( fgets(msg,100,pf) != NULL))
      {
      printf (%s, s);
      }
```

fprintf Escribe la salida formateada a un flujo.

Sintaxis:

```
  int fprintf (FILE *flujo, const char *formato[, argumento, ...]);
```

Esta función es idéntica a la función printf() con la excepción que printf() escribe en la salida estándar (flujo stdout) y la función fprintf() escribe en la salida especificada (flujo indicado en su primer argumento).

Ejemplo:

```
  /* supone que pf est  asociado con
     un fichero de salida */

  fprintf (pf, esto es una prueba %d %f, 5, 2.3);
```

fputc, putc Escribe un carácter en un flujo.

Sintaxis:

```
int fputc (int c, FILE *flujo);
```

La función fputc() escribe un carácter c en el flujo especificado a partir de la posición actual del fichero y entonces incrementa el indicador de posición del fichero. Aunque c tradicionalmente se declare de tipo int, es convertido por fputc() en unsigned char.

El valor devuelto por fputc() es el valor de número de carácter escrito. Si se produce un error, se devuelve EOF.

Ejemplo:

```
/* supone que pf está  asociado con
   un fichero de salida */

fputc ('a', pf);
```

NOTA: putchar(c); es una macro de la función fputc(c,stdout);

fputs Escribe una cadena de caracteres en un flujo.

Sintaxis:

```
int fputs (const char *s, FILE *flujo);
```

La función fputs() escribe el contenido de la cadena de caracteres apuntada por s en el flujo especificado. El carácter nulo de terminación no se escribe.

La función devuelve 0 cuando tiene éxito, y un valor no nulo bajo condición de error.

Ejemplo:

```
/* supone que pf est  asociado con
   un fichero de salida */

fputs ("abc", pf);
```

fread Lee datos de un flujo. (FICHEROS BINARIOS)

Sintaxis:

```
int fread (void *buf, int tam, int n, FILE *flujo);
```

Lee n elementos de tam bytes cada uno. Devuelve el número de elementos (no bytes) realmente leídos. Si se ha producido un fallo el número devuelto será 0.

Ejemplo:

```
/* supone que pf está asociado con
   un fichero de entrada */

float notas[10];

if ( fread (notas, sizeof (float), 10, pf) == 0)
  {
  puts(" Notas leidas");
```

```
    }
 else
    {
    puts("Error en el fichero");
    }
```

fscanf Ejecuta entrada formateada sobre un flujo.

```
 Sintaxis:
   int fscanf (FILE *flujo, const char *formato[, direccion, ...]);
```

Esta función es idéntica a la función scanf() con la excepción que scanf()lee de la entrada estándar (flujo stdin) y la función fscanf () lee de la entrada especificada (flujo indicado en su primer argumento).
Ejemplo:

```
  int d;
  float f;
  fscanf (pf, "%d %f", &d, &f);
```

fseek Posiciona el puntero de fichero de un flujo.(FICHEROS BINARIOS)

```
 Sintaxis:
   int fseek (FILE *flujo, long desplazamiento, int origen);
```

La función fseek() sitúa el indicador de posición del fichero asociado a flujo de acuerdo con los valores de desplazamiento y origen. Su objetivo principal es soportar operaciones de E/S con acceso directo. El desplazamiento es el número de bytes desde el origen elegido a la posición seleccionada. El valor de origen puede ser:

Origen	Nombre
Comienzo del fichero	SEEK_SET
Posición actual	SEEK_CUR
Final del fichero	SEEK_END

Si se devuelve el valor 0, se supone que fseek() se ha ejecutado correctamente. Un valor distinto de 0 indica fallo.
Ejemplo:

```
/* Ejemplo de uso para calcular el
   tamaño de un fichero */

#include <stdio.h>


long tamanio_fichero (FILE *flujo)
  {
     long posicion_aux, longitud;

     // Guardo la posición antigua
     posicion_actual = ftell (flujo);
     // Me sitúo al final
     fseek (flujo, 0L, SEEK_END);
     // Obtengo la posición actual
     longitud = ftell (flujo);
     // Vuelvo a la posición antigua
     fseek (flujo, posicion_aux, SEEK_SET);
     return longitud;
  }


  int main (void)
  {
     FILE *flujo;

     flujo = fopen ("MIFICH.TXT", "w+");
     fprintf (flujo, "Esto es un test.");
     printf ("El tamaño del fichero MIFICH.TXT es %ld bytes,\n",
             tamanio_fichero (flujo));
     fclose (flujo);
     return 0;
  }
```

ftell Devuelve la posición actual en el fichero.(FICHEROS BINAROS)

Sintaxis:

```
  long ftell (FILE *flujo);
```

Devuelve el valor actual del indicador de posición del fichero para el flujo especificado si tiene éxito o -1L en caso de error.

Ejemplo:

```
  /* ver ejemplo anterior */
```

fwrite Escribe en un flujo.(FICHEROS BINARIOS)

Sintaxis:
```
int fwrite (const void *buf, int tam, int n, FILE *flujo);
```

Escribe n elementos de tam bytes cada uno. Devuelve el número de elementos (no bytes) escritos realmente.

Ejemplo:

```
/* supone que pf está asociado con
   un fichero de salida */

float f = 1.2;

fwrite (&f, sizeof (float), 1, pf);
```

remove Función que borra un fichero.

Sintaxis:
```
int remove (const char *nombre_fichero);
```

La función remove() borra el fichero especificado por nombre_fichero. Devuelve 0 si el fichero ha sido correctamente borrado y -1 si se ha producido un error.

Ejemplo:

```
#include <stdio.h>

int main (void)
{
  char fichero[80];

  printf ("Fichero para borrar:" );
  gets (fichero);

  if (remove (fichero) == 0)
    printf ("Fichero %s borrado.\n", fichero);
  else
    printf ("No se ha podido borrar el fichero %s.\n", fichero);

  return 0;
}
```

rename Renombra un fichero.

Sintaxis:
```
int rename (const char *viejo_nombre, const char *nuevo_nombre);
```

La función rename() cambia el nombre del fichero especificado por viejo_nombre a nuevo_nombre. El nuevo_nombre no debe estar asociado a ningún otro en el directorio de entrada. La función rename() devuelve 0 si tiene éxito y un valor no nulo si se produce un error.

Ejemplo:

```
#include <stdio.h>
```

```
  int main (void)
  {
    char viejo_nombre[80], nuevo_nombre[80];
    printf ("Fichero a renombrar:" );
    gets (viejo_nombre);
    printf ("Nuevo nombre:" );
    gets (nuevo_nombre);

    if (rename (viejo_nombre, nuevo_nombre) == 0)
      printf ("Fichero %s renombrado a %s.\n",
             viejo_nombre, nuevo_nombre);
    else
      printf ("No se ha podido renombrar el fichero %s.\n",    viejo_nombre);
    return 0;
  }
```

rewind Posiciona el puntero de fichero al comienzo del flujo.

```
Sintaxis:
  void rewind (FILE *flujo);
```

La función rewind() mueve el indicador de posición del fichero al principio del flujo especificado. También inicializa los indicadores de error y fin de fichero asociados con flujo. No devuelve valor.

Ejemplo:

```
  void releer (FILE *pf)
  {
    /* lee una vez */
    while (! feof (pf))
      putchar (getc (pf));

    rewind (pf);

    /* leer otra vez */
    while (! feof (pf))
      putchar (getc (pf));
  }
```

tmpfile Abre un fichero temporal en modo binario.

```
 Sintaxis:
   FILE *tmpfile (void);
```

La función tmpfile() abre un fichero temporal en modo binario para lectura y escritura. La función utiliza automáticamente un único nombre de fichero para evitar conflictos con los ficheros existentes. La función devuelve un puntero nulo en caso de fallo. El fichero temporal creado por tmpfile() se elimina automáticamente cuando el fichero es cerrado o cuando el programa termina.

Ejemplo:

```
  FILE *pftemp;
  if ((pftemp = tmpfile ()) == NULL)
    {
      printf ("No se puede abrir el fichero de trabajo temporal.\n");
      exit (1);
    }
```

ORGANIZACIÓN DE FICHEROS

La organización de la información en el fichero es fundamental a la hora de realizar operaciones sobre el mismo. Dependiendo del uso, la estructura y la cantidad de información que contenga un archivo tendremos que diseñar la organización más conveniente.

Conceptos previos:
Un fichero puede estar almacenado en dispositivos direccionables (Discos) o secuenciales (Cintas).

Acceso directo: Puedo posicionarme en cualquier parte del fichero para leer o escribir. Sólo se puede hacer sobre dispositivos direccionables

Acceso secuencial: Sólo puede leer o escribir un registro, si ha leído o escrito el anterior. Se puede realizar tanto en dispositivos direccionable como secuenciales

Tipos de organización de ficheros:

- Secuencial
- Secuencial Indexada
- Secuencial Encadenada
- Directa o Relativa

ORGANIZACIÓN SECUENCIAL

- En un fichero organizado de forma secuencial los registros se almacenan en posiciones consecutivas, la posición de cada registro vendrá dado por el valor del campo clave o identificativo.
- En la organización secuencial, para acceder al registro N hay que haber procesado el registro N-1.
- El orden físico de los registros corresponde con el orden lógico.
- Permite usar tanto soportes direccionables como no direccionables.

Recorrido de un fichero secuencial:

```
F = Abrir ( FICHIRO, Lectura)
Leer ( F, Registro)
MIENTRAS ( No FIN ( F ) )
   Mostrar Registro
   Leer (F, Registro)
FIN-MIENTRAS
Cerrar ( F )
```

Ventajas:

- Rápido acceso al siguiente registros en secuencia lógica.
- Permite un rápido recorrido ordenado del fichero
- Se aprovecha todo el espacio del fichero
- Se pueden utilizar cintas magnéticas (gran capacidad, bajo costo)

Inconvenientes

- Para buscar un registro debemos recorrer gran parte del archivo
- Para añadir o borrar (altas o bajas) un registro debemos crear un nuevo fichero para reorganizar todo el archivo.
- Si el fichero es grande, no es una organización adecuada para procesos interactivos, donde hay muchas consultas, altas, bajas y modificaciones.

Alta de un nuevo registro en un fichero secuencial

```
Fe = Abrir (FANTIGUO, Lectura)
Fs = Abrir (FNUEVO, Escritura)
Leer Reg_Nuevo
Escrito = NO
 Leer ( Fe, Reg_fichero)
MIENTRAS ( No FIN(Fe)  AND ( Escrito = NO ))
        // Si encuentro la posición escribo el registro nuevo
        SI ( Reg_fichero. Clave > Reg_Nuevo )
                Escribir(Fs,Reg_Nuevo)
                Escrito = SI
        FIN-SI
        Escribir (Fs, Reg_fichero)
FIN-MIENTRAS
// Si he llegado al final de fichero y todavía no lo escrito
SI ( Escrito = NO)
   Escribir (Fs, Reg_nuevo);
SINO
   // Escribir el resto del fichero
  MIENTRAS ( No FIN(Fe) )
        Escribir(Fs, Reg_fichero)
        Leer(Fe, Reg_fichero)
  FIN-MIENTRAS
FIN-SI
Cerrar(Fe)
Cerrar(Fs)
```

IMPLEMENTACIÓN EN LENGUAJE C:

Recorrido de un fichero secuencial:

```
main ()
{
  FILE *fent;

 // Intento abrir el fichero
  fent = fopen("ALUMNOS.DAT","rb");
  if ( fent == NULL )
     {
     printf(" Error al abrir el archivo ALUMNOS.DAT ");
     return 1;
     }

  fread ( &ralumno, 1, sizeof (TIPOALUMNO), fent );
  while ( !feof (fent) )
     {
     printf(" %-15s, %3d, %3d, %3d. \n",
                        ralumno.nombre, ralumno.edad, ralumno.curso, ralumno.nota );
     fread( &ralumno, 1, sizeof (TIPOALUMNO), fent );
     }
  // Cierro el fichero
  fclose(fent);
}
```

Alta de un nuevo registro un fichero secuencial

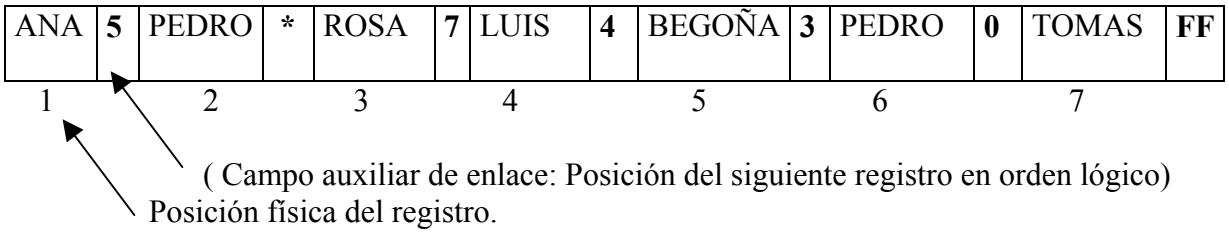
```
main ()

{
  FILE *fent,*fsal;
  TIPOALUMNO ralumno, rnuevo;
  char escrito;

   // Abro el fichero original
  fent = fopen("ALUMNOS.DAT","rb");
  if ( fent  == NULL )
    {
    perror("ALUMNOS.DAT");
    return 1;
    }

  // Creo un fichero temporal
  fsal = fopen("ALUMNOS.TMP","wb");
  if ( fsal  == NULL )
    {
    perror("ALUMNOS.TMP");
    return 2;
    }

  // Leo los datos del nuevo registro a insertar
  LeerCampos(&rnuevo);
```

```
  fread(&ralumno,sizeof(TIPOALUMNO),1,fent );
  escrito = 'n';
  while (  !feof(fent) && escrito == 'n' )
     {
     // Inserto ordenadamente el registro
     if ( strcmp(rnuevo.nombre, ralumno.nombre) < 0)
         {
         // Grabo el nuevo registro
         fwrite(&rnuevo,sizeof(TIPOALUMNO),1,fsal );
         escrito = 's';
         }
     // Grabo el registro antiguo
     fwrite(&ralumno,sizeof(TIPOALUMNO),1,fsal);
     fread (&ralumno,sizeof(TIPOALUMNO),1,fent );
     }

  // Si todavía no el grabado el registro, (Es el último)
  if ( escrito == 'n')
     {
     fwrite(&rnuevo,sizeof(TIPOALUMNO),1,fsal );
     }
  else
     {
     // Escribo el resto del fichero
     while ( !feof(fent) )
         {
         fwrite(&ralumno,sizeof(TIPOALUMNO),1,fsal);
         fread (&ralumno,sizeof(TIPOALUMNO),1,fent );
         }
      }

  // Cierro los ficheros
  fclose(fent);
  fclose(fsal);

  // Borro el fichero original
  remove("ALUMNOS.DAT");

  // Cambio de nombre el fichero auxiliar
  rename("ALUMNOS.TMP","ALUMNOS.DAT");

  puts("Registro insertado.");
}
```

ORGANIZACIÓN SECUENCIAL – ENCADENADA

Los registros se almacenan uno a continuación del otro sin que el orden físico deba corresponder con el orden lógico. Se añade a cada registro un campo auxiliar que señala la posición física del siguiente registro en la secuencia lógica de ordenación.

Ej.-



El valor * significa que el siguiente registro en orden lógico corresponde con el siguiente registro en orden físico. En el ejemplo, el siguiente registro después del registro en la posición 2 con clave PEDRO es el registro 3 con clave ROSA.
El valor FF indica que es el último registro.
El valor 0 indica que el registro está eliminado.

Ventajas:

- Si queremos añadir o incluir un nuevo registro, sólo tenemos que grabarlo al final y reordenar los campos enlaces.
- Si queremos eliminar un registro, lo marcamos como borrado y reordenados los enlaces
- Podemos tener distintas ordenaciones lógicas sin definimos varios campos enlaces en cada registro.

Inconvenientes:

- Se necesita un espacio extra para almacenar los campos enlace
- El soporte debe ser direccionable
- Si el fichero está muy desorganizado, el recorrido en secuencia lógica o la búsqueda de un registro concreto será muy lento.

Recorrido completo de un fichero secuencial-encadenado

```
F =  Abrir (FICHERO, Lectura)
// Se supone que el primer registro físico corresponde con
// el primer registro en orden lógico.
Leer ( F, Registro)
MIENTRAS  (Registro.enlace ≠ FF)
     Mostrar Registro
     // Nos situamos en la posición que indique el campo enlace
     IR_a_posición (F, Registro.enlace)
     Leer(F, Registro)
FIN-MIENTRAS
Mostrar Registro  // Muestro el último
Cerrar(F)
```

Recorrido de un fichero secuencial-encadenado:

```
  FILE *fenc;

  fenc = fopen("ALUMNOSENC.DAT","rb");
  if ( fenc  == NULL )
   {
   perror("ALUMNOSENC.DAT");
   return 1;
   }

  // RECORRIDO DE UN FICHERO SECUENCIAL ENCADENADO
  // Suponemos que debe existir por lo menos un registro y que este
  // debe ser el primero en secuencia lógica.
  fread(&ficha,sizeof(ficha),1,fenc);
  while ( ficha.enlace != -1 )
          {
          printf(" %s\n",ficha.reg.nombre);
          // Si no es el siguiente, voy a la posición indicada
          if ( ficha.enlace != 0 )
            {
            // Voy a la posición indicada por el enlace
            // Resto -1 porque la primera posición en la 0
            fseek ( fenc, (ficha.enlace -1)* sizeof(ficha), SEEK_SET);
            }
          fread(&ficha,sizeof(ficha),1,fenc);
          }
  // Muestro el último.
  printf(" %s\n",ficha.reg.nombre);
   fclose(fenc);
}
```

ORGANIZACIÓN SECUENCIAL – INDEXADA

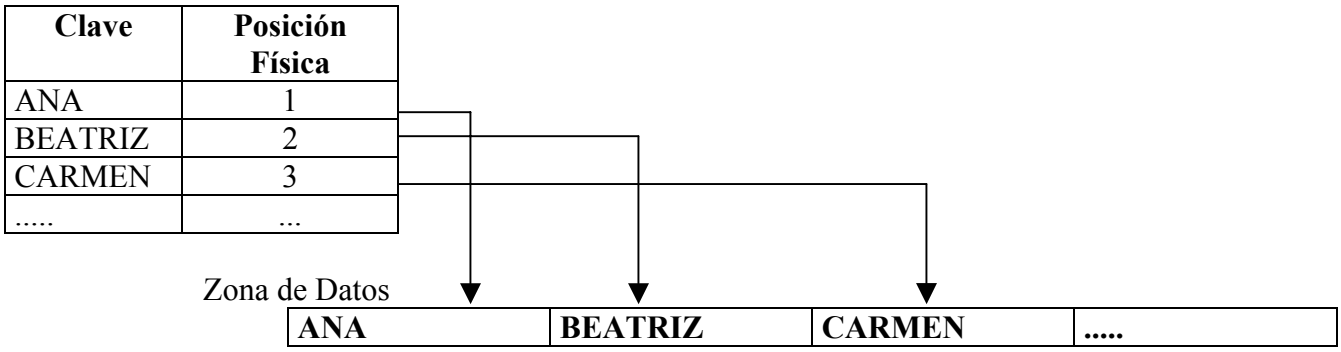
En este método, al igual que la organización secuencial pura, la posición física de los registros coincide con la ordenación lógica. Sin embargo, disponemos de una Zona auxiliar de índices que nos permite una búsqueda dicotómica o binaria de las claves para acceder inmediatamente a la posición física del registro.

- Necesitamos un soporte direccionable

- Para buscar un registro a partir de su clave, primero debemos buscar en la Zona de índices, ir a la posición del registro y leer el mismo.

- El fichero está organizado en dos zonas: Zona de Índices y Zona de Datos.

Zona de Índices:



Podemos tener una entrada en la tabla de índice por cada registro o una entrada por un grupo de registros. Normalmente la zona de índices tiene un tamaño que permite almacenarla en una tabla en la memoria RAM para poder acelerar el proceso de búsqueda.

Ventajas:
- Acceso rápido a cada registro
- Rápido recorrido en secuencia de todo el archivo

Inconvenientes
- Necesita de más espacio, para almacenar la tabla de índices.
- Debemos utilizar un soporte direccionable
- Si queremos añadir o eliminar un registro debemos reorganizar cada vez la zona de índices y la zona de datos.

Para evitar tener que reorganizar el fichero cada vez que realizamos altas y bajas, se suele disponer de una nueva Zona (Zona de Desbordamiento o overflow) donde se sitúan los nuevos registros. Si un registro no está en el índice se busca secuencialmente en la zona de desbordamientos. Para las bajas se elimina la entrada en la Zona de Índices, marcándose el registro como borrado en la zona de datos. Aunque esto evita tener que reorganizar el fichero, conviene que el sistema lo haga de vez en cuando para evitar que la zona overflow sea demasiado grande y se degrade el tiempo de acceso a los registros.

ORGANIZACIÓN DIRECTA O RELATIVA

Los registros se almacenan uno a continuación del otro sin que el orden físico, corresponda con el orden lógico. Para obtener la posición de un registros utilizamos su clave a la que se aplica algún algoritmo. En cualquier caso siempre se necesita un soporte direccionable. Aunque el acceso a un determinado registro puede ser muy rápido el recorrido en secuencia lógica siempre será más lento que el secuencial o el secuencial indexado.
Tres métodos básicos:

Método Directo:

La clave es de tipo numérico y debe corresponder con la posición física dentro del fichero. Si tenemos 1000 registros la clave tiene que estar entre el valor 1 y 1000. Este método es muy rápido pero es demasiado rígido al obligarnos a utilizar un tipo de clave concreto. Por otro lado las altas y la bajas no se tratan adecuadamente.

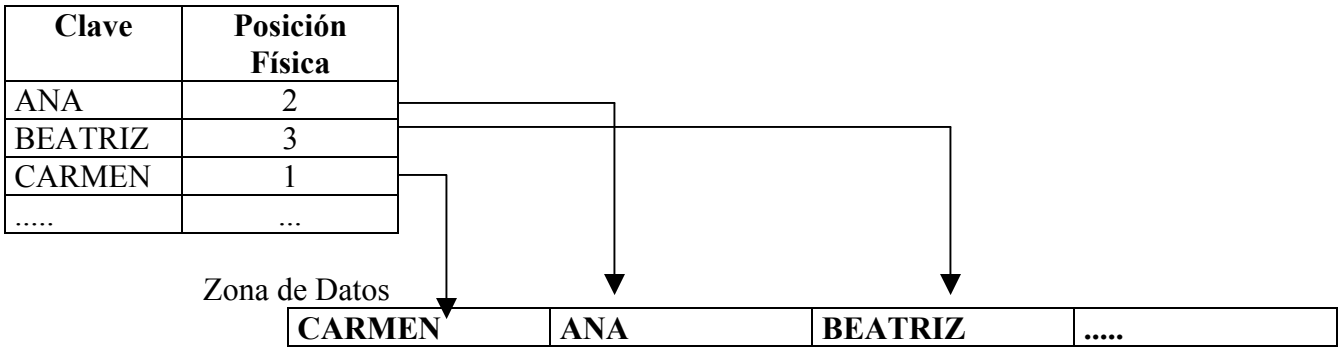
Clave (003) ----------> Posicionarse en el 3º registros del fichero y leer.

Situarse y leer un registro concreto de un fichero:

```
void Otro ( void )
{
  long new_pos; // Nueva posición en bytes
  long max_pos; // Máxima posición en bytes dado el tamaño actual de fichero
  long num_reg; // Número de registro que el usuario quiere leer.

  /* Calculo la posición del ultimo registro */
  fseek(fent,0L,SEEK_END);
  max_pos = ftell(fent) - sizeof(RALUMNO);

  /* Leo una posición y chequeo que es válida */
  do {
       printf(" Nº Reg :"); scanf("%ld", &num_reg);
     /* Posición en bytes a partir del número de registro introducido por el usuario: */
      new_pos = ( num_reg - 1) * sizeof(RALUMNO);
    }
  while ( ( new_pos < 0 ) || ( new_pos > max_pos ) );

  /* Me sitúo y leo el registro indicado */
  fseek (fent, new_pos, SEEK_SET );
  fread (&RALUMNO, sizeof(RALUMNO), 1, fent );
}
```

Método Asociado:

Utiliza una tabla auxiliar como índice donde se asocia el valor de la clave con la posición que ocupa en el fichero. Es similar a la organización secuencial-indexada, aunque en este caso los registros no se encuentran ordenado por la secuencia lógica. Las altas se realizan fácilmente, grabando al final y actualizando el índice. Las bajas implican el borrado de la entrada en la tabla de índices y la colocación de un marca que indica que el registro está borrado. El espacio ocupado por un registro borrado puede ser reutilizado posteriormente por un alta.



Método de transformación de Claves (hash)

Mediante una formula o algoritmo se obtiene a partir de la clave un número, que está entre 1 y N , siendo N el número de registros en la tabla.

Ej.- Suponemos que tenemos una clave alfabética y un fichero con 100 registros. Si la Clave = "ANA" Sumamos el valor de las decimal del ASCII código de la letras:

65 + 78 + 65 = 208 y Calculamos el módulo 208 % 100 = 8

=> La posición del registro es la 8.

El método de transformación de claves es muy rápido a la hora de localizar un registro. Tiene el inconveniente de provocar sinónimos (claves a la que se asigna la misma posición) y huecos (posiciones que no son ocupadas por ningún registro). Existen diversos algoritmos más o menos complejos para evitar estos problemas.

ORDENACIÓN DE UN FICHERO A MEDIANTE UNA TABLA DE REGISTROS

Tablaped Nunped = 5

COD-CLI	COD_ART	FECHA	UNIDADES
CMA003	ART01	12/01/02	9
CMA034	A0034	05/01/02	2
CBA001	A0023	12/01/02	5
CXX001	A0342	08/05/02	8
CAA001	A5400	15/08/02	3
...			
...			

Ordenar tabla

TablaPed (Ordenada)

COD-CLI	COD_ART	FECHA	UNIDADES
CAA001	A5400	15/08/02	3
CBA001	A0023	12/01/02	5
CMA003	ART01	12/01/02	9
CMA034	A0034	05/01/02	2
CXX001	A0342	08/05/02	8
...			
...			

Registro

Cargar fichero en Tabla

FICHERO**: PEDIDOS.DAT**

CMA003 ART01 12/10/23 9	CMA0023 AT003 21/04/02 0	CMA034 A0034 05/01/02 2	CBA001 A00234 12/01/02 5	CXX001 A0342 08/05/02 8	CMA004 A1204 30/01/02 0	CAA001 A5400 15/02/02 3

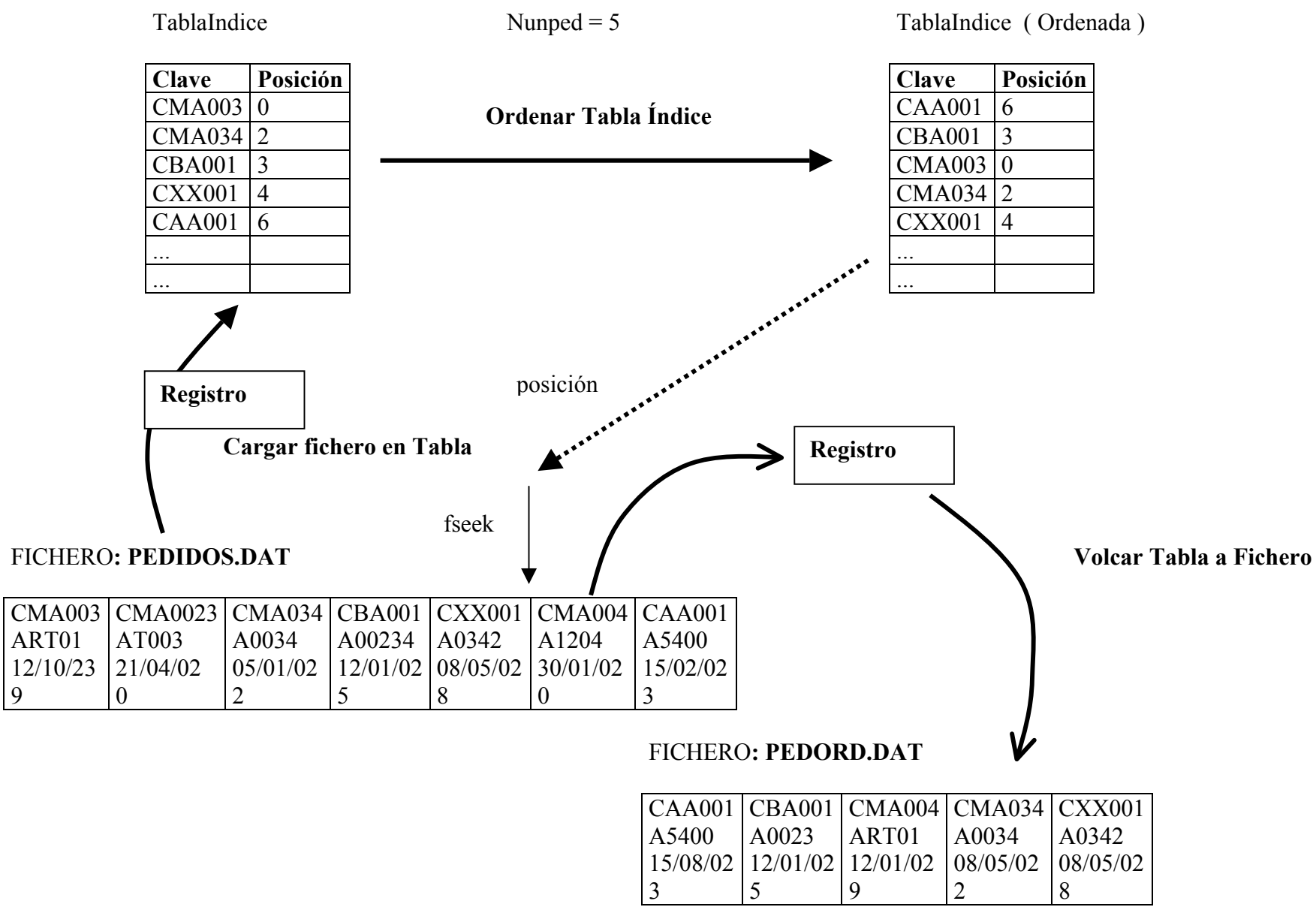
Volcar Tabla a Fichero

Registro

FICHERO**: PEDORD.DAT**

CAA001 A5400 15/08/02 3	CBA001 A0023 12/01/02 5	CMA004 ART01 12/01/02 9	CMA034 A0034 08/05/02 2	CXX001 A0342 08/05/02 8

ORDENACIÓN DE UN FICHERO A MEDIANTE UNA TABLA INDICE



PROBLEMAS SOBRE INFORMES DE FICHERO DE REGISTROS:

Tenemos un archivo de nóminas que contiene registros con la siguiente estructura:

Tipo Nomina

```
  long dni;  // DNI del empleado
  char Nombre[40]:  // Nombre del empleado
  int sueldo; // Sueldo
  int grupo; // Grupo del empleado
  char lugar[20] // Lugar de residencia
```

Realizas un informe agrupado por grupo de empleado de un archivo del nominas con el importe de salarios de los empleados de MADRID. Se supone que el archivo está ordenado por grupo y nombre del empleado.

```
Inicio
Abrir  FNOMINAS para lectura
Total_Madrid = 0
Leer Rempleado
MIENTRAS NO FIN (FNOMINAS)
  grupo_Actual = Rempleado.grupo
  Total_grupo  = 0
  MIENTRAS grupo_Actual = Rempleado.grupo  AND NO FIN(FNOMINAS)
    SI ( Rempleado.lugar = "MADRID")
          Mostrar Datos Rempleado
          Totalgrupo = Totalgrupo + Rempleado.salario
    FINSI
    LEER Rempleado
  FIN-MIENTRAS
  Mostrar Grupo_Actual ,  Total_grupo
  Total = Total + Total_grupo
FIN-MIENTRAS
Mostrar Total_Madrid
Cerrar FNOMINAS
FIN
```

Otros problemas

1.- Mostrar el importe total de las nóminas por cada grupo, se supone que sólo existen 10 grupos, 0 al 9
2.- Mostrar el nombre y el sueldo del empleado con salario más bajo.
3.- Mostrar el nombre y el sueldo de los 5 empleados con salario más alto.
4.- Mostrar el importe total de las nóminas por lugar, se supone que no existen más de 15 lugares distintos.

EJERCICIOS FINAL DE FICHERO:
Se desea realizar el mantenimiento del fichero ARTICULOS.DAT con la siguiente estructura:

```
typedef struct
  {
  char codigo[6];   // Código de artículo   MA034  (5 caracteres máximo )
  short grupo;       // Grupo 1-99
  char  descripcion[30];  // Descripción: Ej.- Martillo  de goma
  char  proveedor[30];    // Nombre del proveedor  Ej- Herramientas Tools S.A.
  int stock;                 // Existencia actual en almacén  >=0
  int stockmin;             // Existencia mínima admisible  >=0
  int preciov;               // Precio de venta >0
  int precioc;               // Precio de compra > 0
  Tfecha  fcompra;       // Fecha de la ultima compra
  Tfecha  fventa;         //  Fecha de la ultima venta
  int   ventasmes[12];  // Tabla con la ventas mensuales
} TipoArticulo;

typedef struct  {short dia;  short mes;  short año;} Tfecha;
```

El programa presenta un registro en la pantalla, mostrando las siguiente opciones:

- Acciones: Nuevo, Borrar, Modificar, Venta, Compra, Terminar

Nuevo(*): Solicita los datos de un nuevo registro grabándolo al final del fichero.
Borrar(***): Realiza un marca lógica mediante un campo auxiliar que indica que el registro esta eliminado, no volverá a mostrarse en la pantalla.
Modificar(**): Solicita nuevos datos para el registro actual, los captura y los graba en el fichero
Venta(*): Solicita la Fecha y cantidad de unidades vendidas del artículo actual, actualizando los datos correspondientes
Compra(*): Solicita la Fecha y la cantidad de unidades compradas del artículo actual, actualizando los datos correspondientes.
Terminar(*): Cierra el fichero y termina el programa. <En el caso de existir registro borrados, se utilizará un fichero auxiliar para guardar los registros válidos, se borrará el fichero antiguo, cambiando de nombre el fichero auxiliar que pasará a ser el nuevo fichero de artículos.>

- Desplazamientos : Siguiente, Anterior, Primero, Ultimo, Otro, Buscar por Código

Siguiente(*): Muestra el siguiente registro del fichero
Anterior(*): Muestra el registro anterior
Ultimo(*) : Muestra el registro último
Otro (*):Se posiciona en un registro determinado
Buscar por código(***): Permite una búsqueda por código de artículo. Se puede realizar una búsqueda secuencial sobre el fichero (solución poco optima) o mantender una tabla índices actualizada, que se cargará al abrir el fichero, sobre la que se realizará una búsqueda binaria o dicotómica.
< Los registros borrados no se deben mostrar en ningún caso>

- Informes:
- 1.Inventario valorado del almacén con subtotales por grupo (*).
- 2.Artículos con stock por debajo del mínimo (*)

- 3.Informe de las ventas mensuales por grupo (**)
- 4.Informe de los 10 artículos más vendidos durante el periodo estival (junio, julio, agosto) (***)

Nota : * Opción obligatoria ** Opción voluntaria simple ***Opción voluntaria compleja

Formato del informe 1.

Nº Pag 1 05/10/2002

INVENTARIO DE ALMACEN

GRUPO	CODIGO	DESCRIPCIÓN	STOCK	P. COMPRA	TOTAL
01					
	A0023	Martillo de goma	23	1000	23000
	B0012	Lápiz Extra	120	100	12000
	C0202	Papel de seda	80	200	1600
				SUBTOTAL GRUPO:	36600

.....

TOTAL ALMACEN 3405300

Formato del informe 2.

Nº Pag 1 05/10/2002

ARTICULO CON STOCK MINIMO

CODIGO	DESCRIPCIÓN	GRUPO	STOCK ACTUAL	STOCK MINIMO
A0023	Martillo de goma	01	10	15
BF034	Teselas metálicas	21	0	6

Formato del informe 3.

Nº Pag 1 05/10/2

VENTAS MENSUALES POR GRUPO

GRUPO	ENE.	FEB.	MAR.	ABR.	MAY.	JUN.	JUL.	AGO.	SEP.	OCT.	NOV	DIC	TOTAL
01	0	34	6	5	2	1	0	0	2	5	0	0	55
02	4	30	3	8	0	3	0	0	4	2	1	10	65
……..													
TOTAL	9394	434	434	4343	3434	56745	565	5656	565	565	90	343	4304590

Formato del informe 4.

Nº Pag 1 05/10/2

ARTICULOS MÁS VEDIDOS (PERIODO ESTIVAL)

CODIGO	DESCRIPCIÓN	GRUPO	P.VENTA	P.COMPRA	JUNIO	JULIO	AGOSTO

A0023	Martillo de goma	01	20	15	2323	120930	83900
BF034	Teselas metálicas	21	10	6	1238	39499	50399
