ESTRUCTURAS DE DATOS EXTERNAS

7.1.- Introducción

La única forma que tiene el ordenador de almacenar información es a través de variables o porciones de la memoria central del mismo. Ésta no es mas que un conjunto de dispositivos electrónicos, con necesidad de alimentación eléctrica, capaces de almacenar datos. Lo que implica que cuando el ordenador se apaga, toda la información que había en la memoria central desaparece.

Por tanto, si se quiere tener acceso a una determinada información, se necesitarán soportes físicos sobre los que depositarla y que sean capaces de contenerla de forma permanente (memorias externas, discos duros, disquetes, etc.).

La forma de guardar los datos en los dispositivos auxiliares es mediante estructuras llamadas ficheros o archivos.

7.2.- Conceptos fundamentales

CAMPO.- Un campo es un conjunto de caracteres capaz de suministrar una determinada información referida a un concepto. Al igual que en las variables, al definir un campo deben indicarse las características:

* **Nombre**: Nombre con el que se identifica ese conjunto de caracteres

* **Tipo**: Tipo de datos que puede contener

* **Tamaño**: Número de caracteres que puede contener el campo.

REGISTRO LÓGICO.- Un registro lógico es un conjunto de campos referentes a una entidad particular.

No debe confundirse en concepto de registro lógico con el de ***REGISTRO FÍSICO*** o ***BLOQUE*** , entendiendo como tal la cantidad de información que puede leerse o escribirse de o en un dispositivo externo en un único acceso es decir, en una única operación de lectura/escritura.

En general, un bloque consta de un número variable de registros lógicos. Esto es, se pueden transferir, en una sola operación de lectura/escritura varios registros lógicos. Este fenómeno se llama ***Bloqueo***, definiéndose como ***Factor de Bloqueo*** al número de registros lógicos contenidos en un bloque. Normalmente, en soportes estándar de almacenamiento el tamaño de un Bloque suele ser 512 bytes

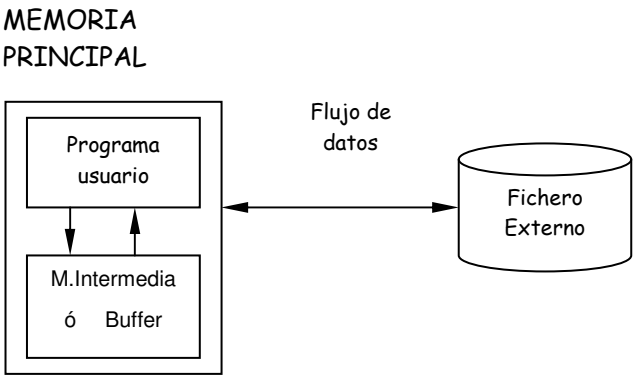
FACTOR DE BLOQUEO.- Es el número de registros lógicos que contiene un registro físico. Puede darse varios casos:

- El Registro lógico es mayor que el Registro físico (registros expandidos).

- El Registro lógico es igual que el Registro físico (registros no bloqueados). El factor de bloqueo es 1.

- El Registro lógico es menor que el Registro físico (registros bloqueados). El factor de bloqueo es mayor que 1.

Para que puedan existir registros bloqueados; es decir, para que en una sola operación de E/S pueda ser transportado más de un registro lógico, será necesario disponer de un área de memoria intermedia que acoja el conjunto de registros que van a ser llevados al fichero desde la memoria principal, en una operación de escritura sobre el fichero o, el conjunto de registros leídos en una operación de lectura del fichero. Esta área de memoria se denomina, en general, buffer.



Por tanto, cuantos más registros lógicos estén contenidos en un registro físico, esto es, cuanto mayor sea el Factor de Bloqueo, menor será el número de accesos necesarios al dispositivo y habrá un mayor aprovechamiento de la capacidad del soporte de almacenamiento.

DIRECCION LOGICA.- (Dirección Software) de un registro es la posición relativa que ocupa en el fichero.

DIRECCION FISICA.- (Dirección Hardware) de un registro es la posición real, física o efectiva donde se encuentra el registro en el soporte de información.

FICHERO o ***ARCHIVO***.- Un fichero es un conjunto de registros homogéneos, almacenados en un soporte externo, que presentan entre sí una relación lógica y que pueden ser consultados individualmente de forma iterativa y sistemática.

Cuando se habla de ficheros, es imprescindible que cada fichero posea un **nombre** para poder identificarlo.

En resumen: Un fichero o archivo está formado por un conjunto de registros y éstos, a su vez, por un conjunto de campos.

7.3.- Organización de un fichero

Se entiende como tal a la forma particular de disponer los registros del mismo en el soporte de almacenamiento durante su creación. Las formas básicas de organizar un fichero son:

7.3.1.- Organización secuencial

Se caracteriza fundamentalmente por almacenar los registros físicamente contiguos en el dispositivo auxiliar (o soporte de almacenamiento), es decir, un registro a continuación de otro y en la misma secuencia en la que se introducen.

7.3.2.- Organización secuencial indexada

Este tipo de organización exige que todos los registros que forman el fichero contengan un campo, denominado ***campo clave***, que servirá para identificar cada registro de forma única, no pudiendo existir en el fichero dos registros distintos con el mismo contenido en dicho campo clave.

En el fichero se almacenan dos tipos de información, los propios registros y los ***índices*** o ***claves*** para llegar a estos registros. Los registros se disponen sobre el soporte en bloques como organización secuencial. Los índices proporcionan la posición de cada uno de estos bloques.

En general, un archivo secuencial indexado constará de tres áreas:

* **Área primaria o área de datos:** Contiene los registros en forma secuencial organizados en secuencia ascendente por la clave (en el momento de su creación) sin dejar huecos libres intercalados. Esta zona está dividida en ***tramos lógicos*** y cada tramo lógico está formado por un conjunto de registros consecutivos.

* **Área de índices:** Es una tabla que contiene información relativa al último y primer registro de cada tramo lógico del área primaria: ***mayor clave del tramo*** (o clave del último registro) y ***dirección del primer registro del tramo***. El área de índices se procesa de forma secuencial.

* **Área de desbordamiento:** Zona reservada para las actualizaciones sobre el fichero secuencial indexado. Sobre esta área los registros se encuentran desordenados, por lo que, si existen muchas actualizaciones, será necesario reorganizar el fichero.

7.3.3.- Organización secuencial encadenada

Cada registro contiene un puntero que permite localizar el siguiente registro sobre el soporte. Es equivalente a una lista enlazada lineal constituida sobre un soporte externo direccionable.

7.3.4.- Organización directa o relativa

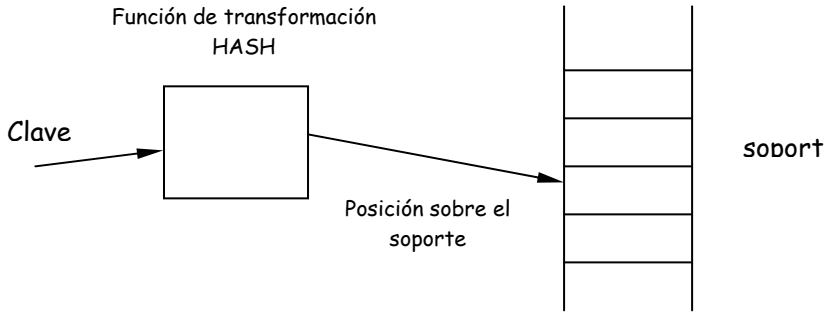
La información almacenada en cada registro ocupa una determinada posición en el fichero que puede conocerse. En estas condiciones los registros pueden leerse o escribirse en cualquier orden, para ello:

* Los registros deben almacenarse sobre un soporte direccionable

* Para un determinado registro, debe existir una correspondencia entre su clave y su posición física. Esta correspondencia la establece el programador y se conoce con el nombre de ***función Hash***.

* El acceso a los registros se realiza mediante la clave a la que se aplica la función Hash correspondiente.

* El acceso a los registros debe ser muy rápido.



7.3.4.1.- La función Hash

La ***función Hash*** debe ser del tipo

Función Hash = f(clave)

Que, al aplicarse sobre el campo clave, genere una dirección relativa al comienzo del archivo sobre el soporte. El objetivo de la función consiste, pues, en generar un número entero traducible a posiciones físicas sobre el soporte. Para ello aplica determinados cálculos aritméticos sobre la clave tales que:

* Los cálculos deben ser rápidos y sencillos para no consumir tiempo de proceso.

* El resultado de los cálculos debe proporcionar direcciones distribuidas uniformemente sobre el soporte, dejando pocos *huecos* sobre éste (posiciones que no se ocupan) y evitando obtener direcciones iguales para claves diferentes (*colisiones*).

* En caso de producirse una colisión, será necesario un tratamiento especial que permita obtener una nueva dirección.

En definitiva, el método Hash consta de dos partes

- Cálculo de la dirección
- Tratamiento de las colisiones

Y tendrá dos usos

* Determinar cómo se almacenan los registros en el fichero

* Determinar cómo acceder a los registros almacenados en el fichero con anterioridad

Entre las funciones Hash mas comunes destacan:

Truncamiento.- Consiste en eliminarlos k primeros dígitos de la clave y los n últimos para una clave numérica de m dígitos: si la clave es 123456789 la función será, supuesto k = 0 y n = 6: f(123456789) = 123

Plegado.- Consiste en dividir en dos o mas partes una determinada clave numérica de n dígitos y sumar (u otra operación) posteriormente dichas partes: si la clave es 123456 la función será, f(123456) = 123 + 456 = 579

Plegado mas truncamiento.- Es una combinación de los métodos anteriores

Direcciones clave.- Asignando a cada valor de la clave una dirección según la relación:

f(clave) = dirección = clave * tamaño_del_registro_lógico

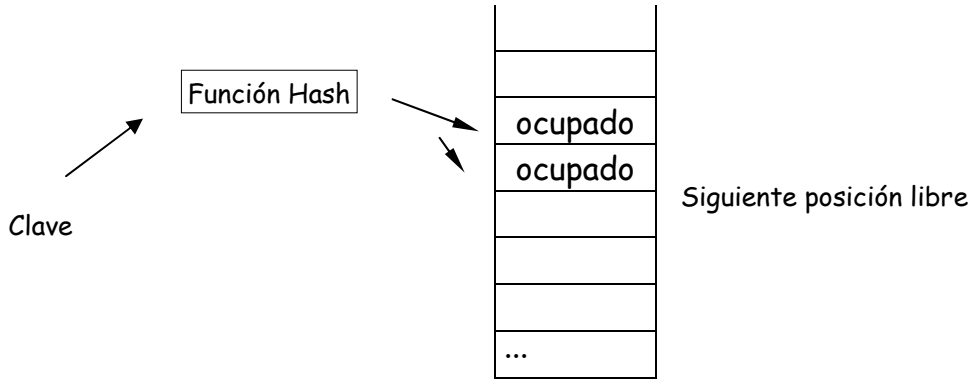
o bién:

f(clave) = dirección = (clave – número de huecos) * tamaño_del_registro_lógico

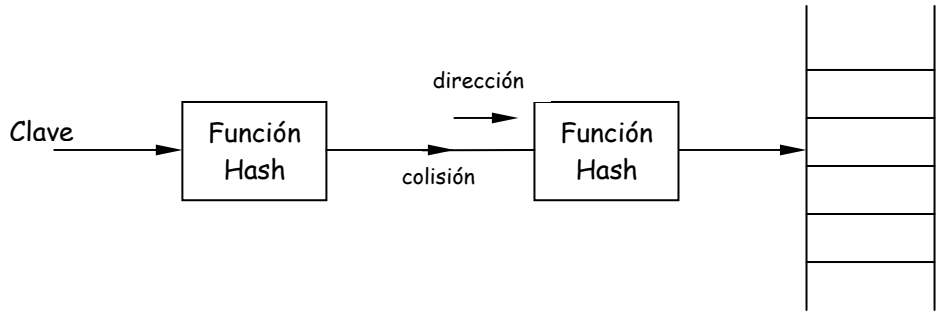
7.3.4.2.- Tratamiento de las colisiones

Una buena función Hash debe minimizar las colisiones ya que el tratamiento de las colisiones suele ser muy complejo. No obstante, si se produce una colisión puede solucionarse de las siguientes formas:

1º.- Buscando la siguiente posición libre en el espacio disponible.



2º.-Aplicando de nuevo, a la dirección obtenida, la misma u otra función Hash.



3º.- Otras formas.

7.4.- Modos de acceso a un fichero

La organización de un fichero y el modo son conceptos completamente diferentes aunque íntimamente relacionados. El modo de acceso es la manera de acceder a los registros de un fichero para extraer (leer) información que pueda ser procesada posteriormente o para grabar información nueva en el fichero. Hay dos modos de acceso:

7.4.1.- Acceso secuencial

Se accede a los registros del fichero según la secuencia física, es decir, uno a continuación de otro en el orden en que fueron escritos. Dicho en otras palabras, para acceder al registro N es preciso paras previamente por los N-1 registros anteriores.

7.4.2.- Acceso directo

Permite el acceso a un registro determinado sin tener que pasar previamente por los registros precedentes.

La utilización de un modo u otro de acceso depende de cómo se organizó el fichero al crearlo, del soporte donde está almacenado y de las necesidades específicas de cada programa.

En los ficheros con organización secuencial el único modo de acceso posible es el acceso secuencial.

7.5.- Clasificación de los registros

Atendiendo a la longitud de los campos que los forman, los registros pueden clasificarse en:

• ***Registros de longitud fija.*** **-** Son aquellos cuya longitud no varía a lo largo del fichero. Atendiendo a su estructura interna, pueden darse las siguientes posibilidades:

- Mismo número de campos por registro e igual tamaño de los campos dentro del registro, para todos los registros

- Mismo número de campos por registro, distinta longitud de los campos dentro del registro y la misma disposición para todos los registros.

- Mismo número de campos por registro, distinta longitud de los campos dentro del registro y distinta disposición para cada registro.

- Diferente número de campos por registro, diferente longitud de los campos dentro del registro y diferente disposición para cada registro.

• ***Registros de longitud variable.-*** Son aquellos cuya longitud varía de un registro a otro. En este tipo de ficheros es preciso establecer, desde el programa que los trate, un método para diferenciar el comienzo y el final de cada campo y cada registro.

7.6.- Clasificación de los ficheros

Atendiendo a su utilización, los ficheros se pueden clasificar en:

- ***Permanentes.-*** Aquellos que sufren pocas alteraciones o variaciones a lo largo del tiempo y contienen información muy valiosa para el funcionamiento de la aplicación que lo utiliza. Pueden ser:

  - ***Maestros o de situación.-*** Encargados de mantener constantemente actualizados los campos cuya información varía.

  - ***Constantes.-*** Contienen información fija y necesaria para el funcionamiento de la aplicación. Tienen un bajo índice de variación en el tiempo.

  - ***Históricos.-*** Contienen información acumulada a lo largo del tiempo sobre las actualizaciones sufridas por los ficheros maestros y constantes o la información de un fichero maestro en un momento determinado.

- ***Temporales.-*** Son aquellos cuya vida está limitada en el tiempo. Pueden ser:

  - ***De Movimiento o Transacción.-*** Contienen la información necesaria para la actualización de ficheros maestros. Su periodo de vida es corto ya que su utilidad termina al actualizarse el fichero maestro.

  - ***De Maniobra o Transitorios.-*** Verdaderos ficheros auxiliares creados durante la ejecución de un programa con el fin de obtener información que será posteriormente procesada. Su periodo de vida es muy corto ya que se destruyen al finalizar la aplicación y no son visibles al usuario.

7.7.- Operaciones con ficheros

Independientemente del tipo de acceso y de organización de un fichero, las operaciones que pueden realizarse en él son:

CREACIÓN

Operación que establece las pautas que determinan la forma en que la información será almacenada en el fichero y será procesada en el futuro, así como el tipo de organización o acceso que se empleará para el manejo de los datos que contenga el fichero.

APERTURA

Para poder utilizar los datos contenidos en un fichero es preciso que éste esté abierto, permitiendo realizar sobre él las operaciones de lectura y escritura necesarias.

Un fichero no debe permanecer abierto mas tiempo que el estrictamente necesario para realizar la operación de lectura o escritura que se realice sobre él.

CIERRE

Finalizadas las operaciones a realizar sobre un fichero, éste debe permanecer cerrado para limitar así l acceso a los datos y evitar un posible deterioro o pérdida de información.

ACTUALIZACIÓN

Operación que permite mantener actualizado el fichero, bien mediante la escritura de nuevos registros, bien mediante la eliminación o modificación de registros ya existentes. La actualización puede afectar a parte o a la totalidad de los campos de un registro.

ORDENACIÓN O CLASIFICACIÓN

Consiste en establecer un orden entre los registros almacenados dentro de un fichero, de acuerdo con criterios previamente definidos. La ordenación puede ser ***ascendente*** o ***descendente***

COPIADO O DUPLICADO

Operación que parte de un fichero origen y crea un fichero nuevo (fichero destino) con la misma estructura y contenido del primero. Dicha operación debe dejar intacto el fichero origen.

CONCATENACIÓN

Partiendo de la existencia de dos ficheros con la misma estructura interna, la operación crea un tercer fichero de igual estructura interna que los anteriores y cuya información es la agregación del contenido del primer y el segundo fichero. La operación no debe afectar a los ficheros originales.

FUSIÓN O MEZCLA

Operación que permite obtener, de dos o mas ficheros con la misma estructura interna y el mismo criterio de clasificación de los datos, un nuevo fichero que contenga los registros de todos los anteriores sin alterar la ordenación que éstos tenían establecida. La operación no debe afectar a los ficheros que intervienen en la fusión.

INTERSECCIÓN

Consiste en crear un nuevo fichero que contiene los registros comunes a dos o mas ficheros con la misma estructura.

PARTICIÓN O ROTURA

Operación que, partiendo de un fichero inicial, obtiene varios ficheros en función de alguna de las características internas de alguno o algunos de sus campos.

COMPACTACIÓN O EMPAQUETAMIENTO

Operación que permite la reorganización de los registros de un fichero eliminando los registros vacíos intermedios (huecos libres), normalmente ocasionados por la eliminación de registros.

CONSULTA

Operación de acceso a registros de un fichero para conocer el contenido de sus campos.

BORRADO O DESTRUCCIÓN

Es la operación inversa a la creación de un fichero y, en consecuencia, una vez efectuada esta operación, el fichero desaparece y se pierde toda posibilidad de acceder a los datos que previamente tenía almacenados.

7.8.- Operaciones con los registros

Con los registros de un fichero, únicamente son posibles las siguientes operaciones:

ALTA

Adición o inserción de uno o varios registros en el fichero. Esta operación sólo es posible si el fichero existe, es decir, fue creado previamente, y está abierto.

BAJA

Eliminación de uno o varios registros de un fichero. La operación requiere un primer proceso de lectura para localizar el registro que se quiere eliminar. La eliminación puede hacerse de dos formas distintas:

1º.- Marcando el registro a eliminar en el fichero mediante el uso de un campo (*bandera o flag*) que no forma parte de los datos. El registro así eliminado sigue permaneciendo en el fichero, pero con acceso limitado, y en consecuencia, sigue ocupando espacio.

2º.- Borrando definitivamente el registro del fichero liberando el espacio ocupado en el soporte externo en el que se encontraba almacenado. Para realizar esta operación, además de una lectura, será preciso un proceso posterior de compactación para eliminar el hueco dejado por el registro.

MODIFICACIÓN

Realización de cambios totales o parciales de uno o varios campos de un registro en un fichero. Requiere una operación previa de lectura para localizar el registro afectado y una operación posterior de escritura para la actualización del registro.

CONSULTA

Operación que permite acceder a uno o varios registros con la intención de visualizar el contenido total o parcial de éstos, ordenados siguiendo criterios de clasificación establecidos por el usuario.

7.9.- Ficheros en el lenguaje C

En el lenguaje C todas las operaciones de entrada y salida tiene lugar bien a través de las funciones de E/S de la biblioteca estándar o bien mediante funciones de E/S de bajo nivel como llamadas al sistema, etc.

El sistema de E/S de C suministra al programador una interfaz entre el programa y los dispositivos externos; este interfaz o nivel de abstracción entre el programa y el dispositivo real utilizado se denomina ***stream*** o flujo, y el dispositivo real utilizado se le denomina ***archivo***.

El sistema de archivos de C permite utiliazr múltiples dispositivos físicos diferentes, transformándolos en dispositivos lógicos (***streams***). Todos los flujos o streams se comportan de forma similar y, por tanto, pueden utilizarse las mismas funciones sobre ellas (esto es, tratan de forma similar a ficheros asociados a dispositivos físicos diferentes). Existen dos tipos de streams:

- ***Flujos de texto:*** Son ristras de caracteres. El estándar C determina que un flujo de texto está organizado en líneas terminadas por un carácter de salto de línea (\n). En un flujo de texto pueden darse ciertas conversiones de caracteres si el entorno del sistema así lo requiere, por lo que puede no haber una relación uno a uno entre los caracteres que se escriben (o se leen) y los que hay en el dispositivo externo. Además, debido a las posibles conversiones, el número de caracteres escritos (o leídos) puede no coincidir con el número de caracteres que haya en el dispositivo externo

- ***Flujos binarios:*** Son ristras de bytes con una correspondencia uno a uno con los del dispositivo externo; esto es, no se producen conversiones de caracteres. Además, el número de caracteres escritos (o leídos) coincide con el número de caracteres que haya en el dispositivo externo

En C un archivo puede ser cualquier cosa, desde un fichero de disco hasta un terminal o una impresora. Mediante la operación de *Apertura* se asocia un stream a un archivo específico. Una vez el archivo esté abierto puede intercambiarse información entre éste y el programa.

Los archivos son diferentes, un archivo de disco, por ejemplo, es direccionable y admite accesos directos, mientras que un archivo de impresora no. Esto ilustra una situación muy importante sobre el sistema de E/S de C: *todos los streams son iguales, pero todos los archivos no*.

Si el archivo permite ***solicitudes de posición***, la apertura del archivo también inicializa el ***indicador de posición del archivo*** al principio del mismo. A medida que se leen o escriben caracteres de o en el archivo, el indicador de posición se va incrementando, asegurando así la progresión sobre el archivo.

Se puede desasociar un archivo de un flujo (stream) específico con una operación de *Cierre*. Todos los archivos se cierran automáticamente cuando el programa que los trata

finaliza normalmente, bien porque ***main*** devuelve el control al Sistema Operativo o porque se produce una llamada a ***exit***(). Los archivos no se cierran si se produce un error de ejecución o se hace una llamada a ***abort***().

7.10.- Elementos básicos del sistema de archivos

El sistema de archivos de C está compuesto por varias funciones interrelacionadas que se encuentran en el fichero de cabecera **stdio.h** y en la librería estándar de C (**stdlib.h**) que proporciona los prototipos de las funciones de E/S y define tres *tipos* de datos **size_t** y **fpos_t** como una variedad de entero sin signo y **FILE**.

Funciones usuales en el sistema de archivos de C

Nombre	**Función**
fopen( )	Abre un archivo
fclose( )	Cierra un archivo
putc( )	Escribe un carácter en un archivo
fputc( )	Lo mismo que putc( )
getc( )	Lee un carácter de un archivo
fgetc( )	Lo mismo que getc( )
fgets( )	Lee una cadena de un archivo
fputs( )	Escribe una cadena en un archivo
fseek( )	Localiza un byte específico en un archivo
ftell( )	Devuelve la posición actual en el archivo
fprintf( )	Hace lo mismo sobre archivos que printf( ) sobre la consola
fscanf( )	Hace lo mismo sobre archivos que scanf( ) sobre la consola
feof( )	Devuelve cierto si se ha llegado al final del fichero
ferror( )	Devuelve cierto si se ha producido un error
rewind( )	Coloca el indicador de posición del archivo al principio del mismo
remove( )	Elimina un archivo
fflush( )	Vacía un archivo

En **stdio.h** también se definen varias macros, las mas relevantes, en el tema actual, son **NULL, EOF, FOPEN_MAX, SEEK_SET, SEEK_CUR** y **SEEK_END**. La macro **NULL**

define un puntero nulo. La macro **EOF,** a menudo definida como -1, es el valor devuelto cuando una función de entrada intenta leer pasado el final del fichero; **FOPEN_MAX** especifica el número máximo de ficheros que pueden estar abiertos simultáneamente. Las macros **SEEK** se utilizan con la función **fsek**(), que es la función que realiza el acceso directo sobre archivos.

7.10.1.- El puntero a archivo

El puntero a archivo es el hilo conductor que unifica el sistema de E/S de C. El puntero a archivo es un puntero a una estructura de tipo **FILE** y apunta a información que define varias cosas sobre el archivo, incluyendo su nombre, su estado y la posición actual dentro de él. En esencia, el puntero a archivo identifica a un archivo específico y es utilizado por el stream asociado para dirigir el funcionamiento de las funciones de E/S.

Para leer o escribir sobre archivos, los programas tienen que utilizar punteros. Para declarar una variable puntero a archivo se utiliza una instrucción como:

```
FILE *fp;
```

7.11.- Funciones del sistema de ficheros de C

FUNCIONES PARA TRATAMIENTO DE FICHEROS

7.11.1.- Apertura de un fichero

La función **fopen**( ) abre un stream para que pueda ser utilizado y vincula un archivo con dicho stream. Posteriormente devuelve el puntero al archivo asociado con ese archivo. La función **fopen**( ) tiene el prototipo:

```
FILE*fopen(const char *nombre, const char * modo);
```

donde *nombre* es un puntero a una cadena de caracteres que especifica el nombre válido del archivo (pudiendo incluir la especificación de la ruta) y la cadena que apunta a *modo* determina como se abre el archivo:

Modo	**Significado**
r	Abre un archivo de texto para lectura
w	Abre un archivo de texto para escritura
a	Abre un archivo de texto para añadir información
rb	Abre un archivo binario para lectura
wb	Abre un archivo binario para escritura
ab	Abre un archivo binario para añadir información
r+	Abre un archivo de texto para lectura/escritura

w+	Crea un archivo de texto para lectura/escritura
a+	Abre o crea un archivo de texto en modo lectura/escritura para añadir información
r+b	Abre un archivo binario para lectura/escritura
w+b	Crea un archivo binario para lectura/escritura
a+b	Abre o crea un archivo binario en modo lectura/escritura para añadir información

La función ***fopen***( ) devuelve un puntero a archivo; si se produce un error cuando se está intentando abrir un archivo, ***fopen***( ) devuelve el puntero nulo.

El código que sigue utiliza la función ***fopen***( ) para abrir, a modo escritura, el fichero 'Prueba.txt':

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

void main()
{
FILE *fp;

if ((fp = fopen("prueba.txt", "w")) == NULL)
        {
                printf ("NO SE PUEDE ABRIR EL FICHERO\n");
                exit(1);
        }
}
```

Aunque la mayoría de los modos de archivo se explican pos sí mismos, deben hacerse algunos comentarios: Si el archivo no existe y se abre para operaciones de sólo lectura la función ***fopen***( ) fallará. Si no existe cuando se abre un archivo para añadir datos, éste se crea. Además, cuando se abra un archivo para añadir, todos los nuevos datos escritos se escribirán al final del fichero. Los contenidos originales permanecerán sin cambios. Si cuando se abre un archivo para escritura el archivo no existe, se creará. Si existe, los contenidos del archivo original se destruirán y se creará un nuevo archivo.

7.11.2.- Cierre de un archivo

Para cerrar un archivo se utiliza la función ***fclose***( ), cuyo prototipo es:

```
int fclose(FILE *fp)
```

La función ***fclose***( ) cierra el archivo asociado a fp, que debe ser un puntero a archivo válido, obtenido previamente utilizando ***fopen***( ), y desliga el flujo del archivo. Para mejorar la eficacia, la mayoría de las implementaciones de sistema de archivos graban los datos en disco sector por sector. Ello implica que los datos están en un *buffer* hasta que se

haya mostrado un sector considerable de información antes de que el *buffer* se escriba físicamente en el disco.

Cuando se llama a la función ***fclose***( ), escribe automáticamente en el disco cualquier información sobrante en un *buffer* parcialmente lleno. A esto se le conoce como ***volcado del buffer***.

Nunca se debe llamar a ***fclose***( ) con un argumento no válido. Al hacerlo se dañará el sistema de archivos y probablemente se produzca una pérdida de datos irrecuperable.

La función ***fclose***( ) devuelve 0 si se ejecuta correctamente. Si se produce algún error devuelve EOF.

ACCESO SECUENCIAL

FUNCIONES PARA ARCHIVOS DE TEXTO

7.11.2.- Funciones de escritura y lectura de un carácter en un fichero

Una vez se ha abierto un fichero, dependiendo del modo de abrirlo, se pueden leer y/o escribir bytes hacia o desde el archivo utilizando las funciones: la función ***fgetc***( ) y la función ***fputc***( ).

La función ***fgetc***( ) y la función equivalente ***getc***( )) lee caracteres del archivo abierto a modo lectura mediante ***fopen***( ), con el siguiente prototipo:

int ***fgetc***(FILE *fp);

donde fp es un puntero al archivo de tipo FILE devuelto por ***fopen***( ).

fgetc( ) devuelve un entero, estando el carácter en el byte menos significativo. A menos que se produzca un error, el byte (o bytes) mas significativo es cero.

La función ***fgetc***( ) devuelve EOF cuando se ha alcanzado el final del fichero, razón por la cual, para leer un fichero hasta el final suele utilizarse el siguiente código:

```
do
{
c = fgetc(fp);
}
while (c != EOF);
```

Debe tenerse en cuenta que, en el caso de producirse un error, ***fgetc***( ) también devuelve EOF, por lo que es conveniente utilizar la función ***ferror***( ) para determinar lo que realmente ha ocurrido.

Por su parte, la función ***fputc***( ) (y su equivalente ***putc***( )) escribe caracteres en un archivo que haya sido abierto previamente para operaciones de escritura con una función ***fopen***( ). El prototipo de la función es:

int ***fputc***(int c, FILE *fp)

donde fp es el puntero al archivo devuelto por ***fopen***( ) y c es el carácter a escribir. El puntero al archivo le dice a ***fputc***( ) en qué archivo de disco se debe escribir. Por razones históricas, c se define como entero, pero sólo se utiliza el byte menos significativo.

Si la operación de ***fputc***( ) tiene éxito, ésta devuelve el carácter escrito. En caso de error devuelve EOF.

El siguiente programa es un ejemplo sencillo que ilustra el uso de las funciones descritas ***fopen***( ), ***fclose***( ), ***fgetc***( ) y ***fputc***( ):

```
/* Programa que muestra la apertura y el cierre de un fichero (MIFICH.TXT)
así como funciones de escritura y lectura de caracteres. Se abre el fichero;
se escribe en el fichero la frase "Es una prueba del sistema de ficheros", se
cierra el fichero y posteriormente se vuelve a abrir para leer el contenido,
presentándolo en la pantalla y, por último, cierra el fichero */

#include <stdio.h>
#include <stdlib.h>

char cadena[80] = "Es una prueba del sistema de ficheros";

void main()
{
        FILE *fp; //definición del flujo o stream asociado al fichero
        char *p;   //puntero a la cadena de caracteres
  char ch;   // variable para determinar el fin del fichero
        int i;      //contador

        /* Abre el fichero MIFICH.TXT para escritura */
        if ((fp = fopen("mifich.txt", "w")) == NULL)
                {
                        printf ("no se puede abrir el fichero\n");
                        exit(1);
    }

        /* Se escribe la cadena en el fichero */
        p = cadena;

        while (*p)
                {
                        if (fputc(*p, fp) == EOF)
                                {
                                        printf ("Error de escritura en el fichero\n");
                                        exit(1);
                                }
                        p++;
                }
        fclose(fp);
```

```
    /* Se abre ahora el fichero a modo lectura */
        if ((fp = fopen("c:\datos\mifich.txt", "r")) == NULL)
                {
                        printf ("no se puede abrir el fichero\n");
                        exit(1);
      }

        /* Se presenta en la pantalla el contenido del fichero */
    printf ("Fichero creado y se pasa a escribir el contenido\n\n");
        for ( ; ; )
                {
                        ch = fgetc(fp);
                        if (ch == EOF) break;
                        putchar(ch);
                }

        /* Cierre del fichero */
        fclose(fp);

        printf ("\n\n\nF I N    D E    P R O G R A M A");

}
```

7.11.3.- Funciones feof( ) y ferror( )

La función ***feof***( ) tiene por fin determinar cuando se ha alcanzado el final del fichero. La función tiene por prototipo:

int ***feof***(FILE *fp);

feof( ) devuelve cierto si se ha alcanzado el final del fichero; en caso contrario devuelve 0. Por esto, la siguiente rutina lee un archivo, tanto de texto como binario, hasta que se encuentra el final del mismo:

```
while (! feof(fp) )   car = fgetc(fp);
```

Este método es normalmente mejor que comprobar carácter por carácter si se llega a EOF. Sin embargo no proporciona ninguna comprobación de errores. Para realizar una comprobación de errores se utiliza la función ***ferror***( ) según:

```
FILE *fp;
… … …
while ( ! feof(fp))
{
        fgetc(cp);
        if (ferror(fp))
        {
                printf("Error de archivo\n");
                break;
        }
}
```

Un ejemplo que ilustra la utilización de estas funciones es el siguiente, que, naturalmente, exige la existencia previa del archivo origen:

```
/* Programa que copia cualquier tipo de fichero, binario o de texto, siendo
el primero ell archivo origen y el segundo el archivo destino. Si el archivo
destino no existe se crea. El programa conlleva comprobación de errores */

#include <stdio.h>
#include <stdlib.h>

void main()
{
FILE *origen, *destino;
char caracter;

// Abrir el archivo origen

if ((origen = fopen("fuente.bin","rb")) == NULL)
        {
                printf ("No se puede abrir el fichero fuente \n");
                exit (1);
        }

//Abrir el archivo destino

if ((destino = fopen("destino.bin","wb")) == NULL)
  {
                printf ("No se puede abrir el fichero destino \n");
                exit (1);
        }

// Copiar un archivo en otro

while(!feof(origen))
        {
  // Lectura de caracteres del fichero origen
                caracter = fgetc(origen);

    // Control de errores en la lectura del fichero origen
                if (ferror(origen))
                        {
                                printf ("Error leyendo el archivo origen\n");
                                exit(1);
                        }

    // Escritura en el fichero destino
                if (!feof(origen)) fputc(caracter, destino);

    // Controil de errores en la escritura del fichero destino
                if (ferror(destino))
```

```
                        {
                                printf ("Error escribiendo en el archivo destino \n");
                                exit(1);
                        }
                }
        fclose(origen);
        fclose(destino);
}
```

7.11.4.- Funciones de mas alto nivel: utilización de cadenas

Cuando se trabaja con archivos de texto C proporciona cuatro funciones que facilitan las operaciones de archivos. Las funciones ***fgets***( ) y ***fputs***( ) que leen y escriben cadenas de caracteres sobre archivos de disco. Estas funciones trabajan como ***fgetc***( ) y ***fputc***( ) respectivamente, pero en lugar de leer o escribir un sólo carácter, leen o escriben cadenas.

Los prototipos de estas funciones son:

int ***fputs*** (cons char **cad,* FILE **fp*);

char ****fgets*** (char **cad,* int *longitud*,FILE **fp*);

La función ***fputs***( ) escribe la cadena apuntada por *cad* en el stream especificado. Si se produce un error devuelve EOF.

La función ***fgets***( ) lee una cadena del flujo especificado hasta que llega a un carácter de salto de línea o se hayan leído *longitud*– 1 caracteres. Si se lee un carácter de salto de línea, éste formará parte de la cadena, a diferencia de la función ***gets***( ). La cadena resultante terminará en nulo. La función devuelve *cad* si se ha ejecutado correctamente y un puntero nulo si se produce un error.

El siguiente programa ilustra la utilización de estas funciones:

```
/* Programa que muestra la utilización de fgets() y fputs() leyendo una cadena del
teclado y escribiéndola en un fichero llamado Prueba.txt. */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main ()
{

char cadena[80];
FILE *fp;

//Apertura del fichero Prueba.txt y creación si no existe

if ((fp = fopen("prueba.txt","w")) == NULL)
        {
                printf ("No se puede abrir el fichero\n");
                exit(1);
```

```
        }

    //Introducción de la cadena por el teclado
    do
        {
            printf("Introducir una cadena y pulsar INTRO para terminar: \n");
            gets(cadena);

    // Se le añade a la cadena un salto de línea
            strcat(cadena,"\n");

    // Se escribe la cadena en el fichero abierto
            fputs(cadena, fp);
            }
    while (*cadena != '\n');
    return (0);
    }
```

El sistema de ficheros de C contiene también otras dos funciones muy potentes: ***fprintf***( ) y ***fcanf***( ), similares a las ya conocidas ***printf***( ) y ***scanf***( ) y que funcionan exactamente igual que ellas, excepto que trabajan con archivos. Sus prototipos son:

int ***fprinf*** (FILE ** *fp,* const char **cadena_de_control*, ...);

int ***fscanf*** (FILE ** *fp,* const char **cadena_de_control*, ...);

donde *fp* es un puntero a archivo devuelto por una llamada a ***fopen***( ).

fprintf( ) y ***fscanf***( ) dirigen sus operaciones de E/S al archivo al que apunta *fp*.

El programa siguiente ilustra la utilización de estas dos funciones:

```
/* Ejemplo de utilización de las funciones fscanf( ) y fprintf( ). El programa
lee una cadena y un entero desde el teclado y los escribe en un archivo de
disco llamado prueba.txt. Posteriormente el programa lee el archivo y
muestra la información en la pantalla*/
#include <stdio.h>
#include <io.h>
#include <stdlib.h>

void main()
{

    FILE *fp;
    char cadena[80];
    int valor;

// Apertura del fichero a modo escritura

    if ((fp=fopen("prueba.txt","w"))==NULL)
```

```
                {
                        printf("No se puede abrir el archivo \n");
                        exit(1);
                }

        // Se lee del teclado
                printf ("Escriba una cadena y un número: ");
                fscanf(stdin, "%s%d", cadena, &valor);

        // Se escribe en el archivo
                fprintf(fp, "%s %d", cadena, valor);

        // Cierre del fichero
                fclose(fp);

        // Apertura del fichero para ver el contenido
                if ((fp=fopen("prueba.txt","r"))==NULL)
                        {
                                printf("No se puede abrir el archivo \n");
                                exit(1);
                        }

        // Lectura del contenido del fichero
                fscanf(fp, "%s%d", cadena, &valor);

        // Y presentación en la pantalla
                fprintf(stdout,"%s %d", cadena, valor);

        }
```

7.11.5.- La función rewind( )

La función rewind ( ) reinicializa el indicador de posición al principio del archivo especificado por su argumento. Esto es, "rebobina" el fichero. Su prototipo es:

void **rewind**(FILE *fp)

donde fp es un puntero a fichero válido.

Un ejemplo de función rewind está en el programa:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main ()
{
char cadena[80];
FILE *fp;
//Apertura del fichero Prueba.txt y creación si no existe
if ((fp = fopen("prueba.txt","w+")) == NULL)
        {
                printf ("No se puede abrir el fichero\n");
```

```
                exit(1);
        }
//Introducción de la cadena por el teclado
do
        {
                printf("Introducir una cadena y pulsar INTRO para terminar: \n");
                gets(cadena);
// Se le añade a la cadena un salto de línea
                strcat(cadena,"\n");
// Se escribe la cadena en el fichero abierto
                fputs(cadena, fp);
                }
while (*cadena != '\n');
// Ahora lee y muestra el fichero otra vez
// reinicializando el indicador de posición del archivo al inicio del mismo
rewind(fp);
while (!feof(fp))
{
        fgets(cadena,79,fp);
        printf (cadena);
}
printf ("\n\nF I N      D E     P R O G R A M A");
return (0);
}
```

7.11.6.- Eliminación de archivos

La función **remove( )** elimina el archivo especificado. Su prototipo es:

int **remove** (const char *nombre)

La función devuelve 0 si tiene éxito. En caso contrario devuelve un valor distinto de 0.

Un ejemplo de eliinación de archivos es:

```
/* Eliminación de un fichero */
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int main()
{
char cadena[80];
char Nombre[24]= "prueba.txt";

printf("¿eliminar %s (S/N)",Nombre);
gets (cadena);
if (toupper(*cadena) == 'S')
// Utilización de la función remove
        if (remove(Nombre))
       {
```

```
                    printf("No se puede eliminar el fichero.\n");
                    exit(1);
                    }
        printf ("F I N    D E    P R O G R A M A");

        return (0);
}
```

7.11.7.- Volcado de un flujo

Si se desea vaciar el contenido de una secuencia de salida, se utiliza **fflush**( ) cuyo prototipo es el siguiente:

int **fflush**(FILE *fp)

Esta función escribe los datos del búfer en el archivo asociado con fp. Si se llama a **fflush** ( ) con un puntero nulo, se vacian los buferes de todos los archivos abiertos para salida.

La función **fflush**( ) devuelve 0 si tiene éxito; en caso contrario devuelve EOF.

FUNCIONES PARA ARCHIVOS BINARIOS

7.12.- Las funciones fread y fwrite

Para leer y escribir tipos de datos que ocupen mas de un byte, el sistema de archivos de C proporciona las funciones **fread( )** y **fwrite**( ). Estas funciones permiten la lectura y escritura de bloques de cualquier tipo de datos. Sus prototipos son:

size_t **fread**(void *bufer, size_t num_bytes, size_t cuenta, FILE *fp);

size_t **fwrite(**const void *bufer, size_t num_bytes, size_t cuenta, FILE *fp);

En fread, bufer es un puntero a la región de memoria que recibirá los datos del archivo. En **fwrite**( ), bufer es un puntero a la información que se va a escribir en el archivo. El valor de cuenta determina cuántos elementos se van a leer o escribir, cada uno de ellos de num_bytes de longitud y, por último, fp es un puntero al archivo de una secuencia abierta previamente.

La función **fread**( ) devuelve el número de elementos leídos. Este valor }puede ser menor que cuenta si se ha encontrado el final del fichero o se ha producido un error.

La función **fwrite**( ) devuelve el número de elementos escritos . Este valor será igual a cuenta salvo que se produzca un error.

7.12.1.- Uso de fread( ) y fwrite( )

Siempre que se haya abierto el fichero como binario, **fread**( ) y **fwrite**( ) pueden leer y escribir cualquier tipo de información. Un ejemplo es el programa siguiente:

```
/* Escribe datos en un fichero que NO son de tipo caracter y los lee
posteriormente */
```

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

int main ()
{
        FILE *fp;
        double d = 12.23;
        int i = 101;
        long l = 123023L;

        if ((fp=fopen("prueba.txt","wb+"))==NULL)
                {
                        printf ("NO se puede abrir el fichero");
                        exit(1);
                }

        fwrite(&d, sizeof(double), 1, fp);
        fwrite(&i, sizeof(int), 1, fp);
        fwrite(&l, sizeof(long), 1, fp);

        rewind(fp);

        fread(&d, sizeof(double), 1, fp);
        fread(&i, sizeof(int), 1, fp);
        fread(&l, sizeof(long), 1, fp);

        printf("%f    %d    %ld ",d, i, l);

        fclose(fp);

  printf("\n\n    F I N    D E    P R O G R A M A");
        return (0);
        }
```

Como se observa, el buffer puede ser (y normalmente es) simplemente la memoria utilizada para almacenar una variable; además, aunque se ha ignorado el valor devuelto por las funciones, estos valores deben comprobarse por si se producen errores.

Una de las aplicaciones mas útiles de las funciones fread y fwrite es la lectura y escritura de tipos de datos definidos por el usuario, especialmente estructuras. Por ejemplo, la estructura:

```
struct tipo_struct
{
float valor;
char nombre[80];
} ficha;
```

la siguiente instrucción escribe el contenido de ficha en el archivo apuntado por fp:

fwrite (&ficha, sizeof(struct tipo_struct), 1, fp);

ACCESO DIRECTO A ARCHIVOS EN C

7.13.- fseek y E/S de acceso directo

Utilizando el sistema de E/S de C se pueden realizar operaciones de lectura y escritura directas mediante la función **fseek()**, que sitúa el indicador de posición del archivo. Su prototipo es el siguiente:

int fseek (**FILE** *fp, **long** numbytes, **int** origen)

donde fp es un puntero dirigido al fichero que se desea acceder; numbytes es el número de bytes necesarios, a partir del origen, para acceder a la posición deseada y origen es una de las siguientes macros:

MACRO	ORIGEN	VALOR
SEEK_SET	Desde el principio del fichero	0
SEEK_CUR	Desde la posición actual	1
SEEK_END	Desde el final del fichero	2

Estas macros corresponden a valores enteros e indican desde qué posición se comienza a sumar nubytes para hallar la posición final.

Normalmente se utiliza fseek () con ficheros binarios, pues en su tratamiento no hay conversión de datos. La función devuelve 0 cuando no hay errores; en caso contrario devuelve un número diferente de 0.

El programa siguiente ilustra la utilización de fseek(). Busca y muestra el byte especificado en el archivo que se indique. En la línea de órdenes se debe especificar el nombre del archivo y después el byte a buscar.

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

int main (int argc, char *argv[])
{
        FILE *fp;
   if (argc != 3)
   {
        printf ("Uso: BUSCAR archivo byte \n");
       exit(1);
   }
        if ((fp = fopen(argv[1],"r")) == NULL)
   {
        printf ("No se puede abrir el archivo.\n");
       exit(1);
   }
   if (fseek(fp, atol(argv[2]), SEEK_SET)
   {
```

```
            printf ("Error de posicionamiento.\n");
            exit(1);
        }
        printf ("El byte %ld es %c.\n", atol(argv[2]), getc(fp));
        fclose(fp);
        return 0;
    }
```

Puede determinarse la posición actual en un archivo utilizando la función **ftell( )**. Su prototipo es:

Long int ftell(FILE *fp)

Que devuelve la ubicación de la posición actual en el archivo asociado a fp. Si se produce algún fallo devuelve -1.

7.14.- Conexión entre la consola y la E/S; Reenvío de la E/S

C distingue poco entre la E/S por consola y la E/S por archivos. Las funciones de E/S descritas anteriormente dirigen sus operaciones a **stdin** o **stdout.** En esencia, las funciones de E/S por consola son sencillamente versiones especiales de las funciones análogas para archivos. La razón de su existencia es que resultan convenientes para el programador.

En entornos en los que se permite la redirección de la E/S puede hacerse que stdin y stdout sean asignados a otros dispositivos aparte del teclado y la pantalla. Por ejemplo, si se considera el programa:

```
#include <stdio.h>
int main()
{
char cad[80];
printf ("Introduce una cadena:   ");
gets (cad);
printf (cad);
}
```

Y se supone que al programa anterior se le llama PRUEBA.

Si se ejecuta PRUEBA normalmente mostrará su petición en la pantalla.

La Entrada/Salida perteneciente a las funciones que se refieren exclusivamente a la consola del ordenador, se pueden reenviar a otros dispositivos. Para ello se emplean los operadores de reenvío:

> para reenviar la salida

< para reenviar la entrada

Por ejemplo, si desde la línea de órdenes del sistema operativo se escribe:

PRUEBA > SALIDA

Se hace que la salida de PRUEBA se reescriba a un fichero llamado SALIDA. Además, si se escribe:

PRUEBA >> SALIDA

La salida de PRUEBA se reenvía al final del fichero SALIDA

Análogamente, si se escribe

PRUEBA < ENTRADA

La entrada al programa PRUEBA se obtiene desde el fichero ENTRADA. Todas las lecturas dirigidas a la consola se realizarán desde el fichero ENTRADA.

Para reenviar tanto la entrada como la salida se combinan ambas formas:

PRUEBA < ENTRADA > SALIDA

Redireccionando stdin a un archivo llamado ENTRADA y se envía la salida a un archivo llamado SALIDA.

7.14.1.- Uso de freopen() para redirigir las secuencias estándar

Se pueden redirigir las secuencias estándar utilizando la función **freopen()**. Esta función asocia una secuencia con un archivo nuevo. Por tanto, se puede utilizar para asociar una secuencia estándar con un archivo nuevo. So prototipo es:

FILE *freopen(const char **nombre*, const char * *modo*, FILE * *secuencia*);

Donde *nombre* es un puntero al nombre del archivo que se quiere asociar con la secuencia apuntada por *secuencia*. El archivo se abre de acuerdo con el valor de *modo*, que puede tomar los mismos valores que para **fopen()**. **freopen()** devuelve *secuencia* si tiene éxito y **NULL** si se produce algún error.

El siguiente programa utiliza **freopen()** para redirigir **stdout** a un archivo llamado SALIDA:

```
#include <stdio.h>
int main()
{
char cad[80];
freopen("SALIDA", "w", stdout);
printf ("Introducir una cadena de caracteres: ");
gets (cad);
printf (cad);
return 0;
}
```

En general, resulta útil redirigir las secuencias estándar en situaciones especiales, como al depurar, utilizando **freopen()**. Sin embargo, es menos eficiente la E/S sobre disco con **stdin** y **stdout** redirigidos que utilizando las funciones **fread()** y **fwrite()**.

7.15.- Ejemplos

```
/* programa que cuenta los caracteres, palabras y lineas de un fichero
que se recibe de arguento. La apertura del fichero se realiza en modo texto y
solo lectura */
#include <stdio.h>

#define SI 1
#define NO 0
#define BOOL int

int separador (char ch);

void main (int argc, char *argv[])
{
        FILE *fp;
    int ncar = 0; //Contador de caracteres
    int npal = 0; //Contador de palabras
    int nlin = 0; //Contador de lineas
    BOOL enpalabra = NO; // dentro de una palabra
    char ch;

    if (!(fp = fopen(argv[1], "r")))
        {
        printf ("Error al abrir el fichero %s", argv[1]);
          exit (1);
       }

    while ((ch =getc(fp)) != EOF)
        {
        ncar++;
          if(ch == '\n') nlin++;
          if (separador (ch))
                {
                if (enpalabra)
                        {
                        enpalabra = NO;
                        npal++;
                    }
              }
          else if (!enpalabra) enpalabra = SI;
       }

    printf ("\n\nFichero:    %s\n",argv[1]);
    printf ("Número de caracteres: %d\n",ncar);
    printf ("Número de palabras: %d\n",npal);
    printf ("Númewro de líneas: %d\n",nlin);

    fclose(fp);
}
```

```
int separador (char ch)
{
        if (ch = ' ' || ch == '\n' || ch == '\t') return 1;
    return 0;
}


/* programa que vuelca en pantalla el contenido de un fichero de texto,
que recibe como argumento */
#include <stdio.h>
void main (int argc, char *argv[])
{
FILE *fp;
char ch;
if (!(fp = fopen(argv[1], "r")))
        {
        printf ("Error al abrir el fichero %s",argv[1]);
          exit(1);
       }

    do
    {
        putchar (ch = getc(fp));
    }
    while (ch != EOF);
    fclose (fp);
}


/* programa que crea un fichero desde el teclado,en modo texto y sole escritura
cada caracter que se lea del teclado se escribe en el fichero. El bucle termina
cuando se lee EOF*/
#include <stdio.h>

void main (int argc, char *argv[])
{
        FILE *fp;
        char ch;
    if (!(fp = fopen(argv[1], "w")))
        {
        printf ("Error al abrir el fichero %s",argv[1]);
          exit(1);
       }

    do
    {
        putc (ch = getchar(), fp);
    }
    while (ch != EOF);
    fclose (fp);
```

```
}
```

```
/* programa que escribe cadenas de caracteres en cualquier fichero recibiendo
como argumentos el canal asociado al fichero, la cadena de caracteres y el nú-
mero de lineas que se desean saltar una vez escrita la cadena*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void escribirlinea(FILE *fp, char *s, int nlineas);

void main (void)
{
        escribirlinea (stdout, "una linea", 0);
        escribirlinea (stdout, "otra linea", 1);
        escribirlinea (stdout, "otra linea mas", 2);
        escribirlinea (stdout, "y otra mas", 1);
        escribirlinea (stderr, "escribe una linea en el canal de errores", 2);
        escribirlinea (stdprn, "escribe una linea en la impresora", 3);
}


void escribirlinea(FILE *fp, char *s, int nlineas)
{
int i;
while (*s)
        fputc(*s++,fp);

for (i = 0; i < nlineas; i++)
        putc ('\n',fp);
}
```

```
/* programa que copia un fichero de texto sobre otro, leyendo la información
carácter a carácter*/
#include <stdio.h>
void main (argc, *argv[])
{
FILE *fp1, *fp2;
char ch;
if(argc != 3)
        {
        puts ("Número de paráetros erróneo.");
       exit(1);
   }
if (!(fp1 = fopen(argv[1], "r")))
```

```
        {
        printf ("Error al abrir el fichero %s\n",argv[1]);
          exit(1);
       }

if (!(fp2 = fopen(argv[2], "w")))
        {
        printf ("Error al abrir el fichero %s",argv[2]);
          exit(1);
       }

do
   {
        putc (ch = getc(fp1),fp2);
   }
while (ch != EOF);
fclose (fp1);
fclose (fp2);

}
```

```
/* programa que copia un fichero de texto sobre otro, leyendo la información
linea a linea*/
#include <stdio.h>
#include <string.h>

void main (argc, *argv[])
{
FILE *fp1, *fp2;
char bufertemp[80];

  if(argc != 3)
        {
        puts ("Número de parámetros erróneo.");
       exit(1);
   }

  if (!(strcmp(argv[1],argv[2])))
        {
        puts ("Los archivos tienen que ser diferentes.");
       exit(1);
   }

  if (!(fp1 = fopen(argv[1], "r")))
        {
        printf ("Error al abrir el fichero %s\n",argv[1]);
          exit(1);
```

```
        }

   if (!(fp2 = fopen(argv[2], "w")))
         {
         printf ("Error al abrir el fichero %s",argv[2]);
           exit(1);
        }

   while (fgets (bufertemp,80,fp1))
            fputs (bufertemp,fp2);

    fclose (fp1);
    fclose (fp2);

}
```

```
/* programa que envia a la impresora los caracteres que lee del teclado
utilizando el canal PRN*/
#include "stdio.h"

void main ()
{
FILE *fp;
char ch;


   if (!(fp = fopen("prn", "w")))
         {
         printf ("Error de conexion");
           exit(1);
        }

   do
   {
         putc (ch = getchar(), fp);
   }
   while (ch != EOF);
   fclose (fp);
}
```

```
/* programa que envia a la impresora el fichero que recibe como argumento */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
```

```
void main (int argc, char *argv[])
{
FILE *fp1, *fp2;
char ch;


 if (!(fp1 = fopen(argv[1], "r")))
        {
        printf ("Error al abrir el fichero %s\n",argv[1]);
          exit(1);
       }

  if (!(fp2 = fopen("prn", "w")))
        {
        printf ("Error de conexion");
          exit(1);
       }

  do
  {
        putc (ch = getc(fp1), fp2);
  }
  while (ch != EOF);
  fclose (fp1);
  fclose (fp2);
}
```

```
/* programa que busca una palabra en un fichero de texto empleando las funciones
fgets()para leer lineas y strstr() para averiguar si la palabra está presente
dentro de una linea*/
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>

void main (int argc, char *argv[])
{
FILE *fp;
char bufertemp[80];
int nlinea = 0;


 if(argc != 3)
        {
        puts ("Número de parámetros erróneo.");
       exit(1);
   }
```

```
 if (!(fp = fopen(argv[2], "r")))
         {
         printf ("Error al abrir el fichero %s\n",argv[2]);
           exit(1);
        }

 while (fgets (bufertemp,80,fp))
         {
         nlinea++;
        if (strstr(bufertemp,argv[1]))
        {
           // la palabra esta presente
         printf ("%s linea %d: ",argv[2], nlinea);
           puts (bufertemp);
        }
    }
 fclose (fp);
}
```

```
/* programa que concatena dos ficheros sobre un tercer fichero. Si no se define
el fichero destino, la salida se escribe en la pantalla. Si el fichero destino
es un asterisco la salida se escribe en la impresora. */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>

void main (int argc, char *argv[])
{
FILE *fp1, *fp2, *fp3;
char ch;

 if (!(fp1 = fopen(argv[1], "r")))
         {
         printf ("Error al abrir el fichero %s\n",argv[1]);
           exit(1);
        }

 if (!(fp2 = fopen(argv[2], "r")))
         {
         printf ("Error al abrir el fichero %s\n",argv[2]);
           exit(1);
        }

 if (!(strcmp(argv[1],argv[3])|| !(strcmp(argv[2],argv[3])))
         {
```

```
        puts ("El fichero destino debe ser diferente de los dos ficheros origen");
       exit(1);
   }


 if (!argv[3]) fp3 = stdout; // Salida por pantalla

 else if (argv[3][0] = '*') fp3 = stdprn;

 else if (!(fp3 = fopen(argv[3], "w")))
        {
        printf ("Error al abrir el fichero %s\n",argv[1]);
          exit(1);
       }

 do
 {
        if (ch = getc(fp1), ch != EOF) putc (ch, fp3);
 }
 while (ch != EOF);

 do
 {
        putc (ch = getc(fp2), fp3);
 }
 while (ch != EOF);
fclose fp1;
fclose fp2;
fclose fp3;
}
```

```
/* programa que concatena dos ficheros sobre un tercer fichero. de forma que al
segundo fichero se le añada el contenido del primero */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>

void main (int argc, char *argv[])
{
FILE *fp1, *fp2;
char bufertemp[80];

 if (argc != 3)
 {
        puts ("Numero de parametros erroneo");
```

```
   exit(1);
 }

 if (!(strcmp(argv[1],argv[2])))
 {
        puts ("Los dos ficheros deben ser diferentes");
   exit (1);
 }


 if (!(fp1 = fopen(argv[1], "r")))
        {
        printf ("Error al abrir el fichero %s\n",argv[1]);
          exit(1);
       }

 if (!(fp2 = fopen(argv[2], "a")))
        {
        printf ("Error al abrir el fichero %s\n",argv[2]);
          exit(1);
       }


 while (fgets(bufertemp, 80, fp1))
                fputs(bufertemp, 80, fp2));

 fclose(fp1);
 fclose(fp2);

}
```

```
/* programa que lee un fichero binario con los datos de una estructura, permite
consultar un registro y visualizarlo en pantalla */
#include <stdio.h>
#include <conio.h>
#include <mem.h>
#include <stdlib.h>
#include <string.h>

#define FOREVER for(;;)
#define LONGCAMPO 40
#define NO 0
#define SI 1

typedef struct
   {
        char nombre[LONGCAMPO];
```

```
        char x;
        char y;
        int z;
        float u;
        char sobran[2];
        }REGISTRO;

    #define LONBUF sizeof (REGISTRO)

    REGISTRO datos;

    int modificar(char *nombre, int *x);

    void main()
    {
            FILE *fp;
        int nbytes = LONBUF;
        int n;

            if (!(fp = fopen("archivo.dat", "r+b")))
            {
            printf ("Error al abrir el fichero archivo.dat");
              exit(1);
           }

           FOREVER
           {
            FOREVER
                    {
                    printf ("\nRegistro a consultar:   ");
                     scanf ("%d",&n);
                     if (n == -1)
                            {
                            fclose (fp);
                            exit(0);
                        }
                     if (n >= 1 && n <= 10) break;
                }
                if (fseek(fp,(long)--n*LONBUF,SEEK_SET))
                  {
                            perror ("Error accediendo al fichero archivo.dat");
                       exit(1);
                     }
                fread (&datos,1,nbytes,fp);
                printf ("\n\nNOMBRE: %s\nOTROS CAMPOS: %5d %5d %5d
%f\n\n",
                                    datos.nombre, datos.x, datos.y, datos.z, datos.u);

                if (modificar (datos.nombre, &datos.x))
                {
```

```
                    if (fseek (fp,(long)n*LONBUF, SEEK_SET))
                     {
                            perror ("Error accediendo al fichero archivo.dat");
                        exit(1);
                     }
                     else
                      fwrite (&datos,1,nbytes,fp);
                  }
          }
    }


    int modificar(char *nombre, int *x)
    {
    char buffer[100];
    int actualizar = NO;
    /* la función permite modificar dos campos del registro */

    fflush(stdin);
    printf ("NUEVO NOMBRE:     ");
    gets (buffer);
    if (buffer[0])
            {
             buffer [LONGCAMPO-1] = '\0';
           strcpy(nombre,buffer);
           actualizar = SI;
      }

    printf ("NUEVO CAMPO X[1..127]:    ");
    gets (buffer);
    if (buffer[0])
            {
             *x = atoi(buffer);
           actualizar = SI;
      }
      return actualizar;
   }
```
