Capítulo 7

Introducción al Uso de Ficheros

Índice General

"Como he dicho, el problema es muy sencillo. Ni siquiera en este edificio podemos crear indefinidamente más ficheros que los que destruímos. Ahora los archivos ya están que revientan."

"Entonces vino Harting y se encargó de ese trabajo."

"En efecto."

John Le Carré. "Una pequeña ciudad de Alemania".

7.1. Introducción

Los programas que se han visto hasta el momento trabajan con datos que se introducen por la entrada estándar (teclado) y cuyos resultados se escriben en la salida estándar (pantalla).

Se ha estudiado cómo almacenar dichos datos en diversas estructuras estáticas o dinámicas (vectores, tuplas, listas, ...), eligiendo siempre la estructura más adecuada para conseguir que el proceso de los datos sea lo más eficiente posible. Pero, hasta ahora, los datos se han almacenado en estructuras internas del ordenador. Y el problema, cuando se actúa de esa forma, es que cuando finaliza el programa se pierden tanto los datos como los resultados que el programa había procesado, puesto que se trabaja con datos residentes en la memoria principal del computador; es decir, en cuanto el programa finaliza su ejecución su zona de datos se libera. Además, la memoria es volátil: si se apaga el computador (o hay un fallo en la corriente eléctrica) su contenido se pierde completamente.

Trabajar exclusivamente con datos residentes en memoria no parece ser una buena forma de trabajo. Si lo que se pretende, por ejemplo, es disponer de una buena estructura de datos en la que almacenar de forma eficiente la información de los alumnos de esta asignatura, no parece una buena política que esos datos deban *reintroducirse completamente* cada vez que se pretenda consultar o procesar una determinada información (buscar la nota de un determinado alumno, consultar cuántos alumnos han aprobado u obtener un listado por orden alfabético de un determinado grupo de prácticas).

En estos casos, se debe disponer de la posibilidad de almacenar los datos de forma permanente en un dispositivo de almacenamiento que *no sea volátil*, es decir, poder almacenar y recuperar los datos en un dispositivo de *almacén secundario*, para que los datos no se pierdan de una ejecución a otra o, simplemente, cuando se apaga el computador. La mayoría de los lenguajes de programación permiten esta posibilidad de almacenar y recuperar los datos en dispositivos de almacenamiento secundario (discos duros, disquetes, CD-ROMs, DVDs, ...) para que no se pierda la información mediante instrucciones de manejo de *ficheros*.

Definición 7.1 (Fichero) *Estructura de datos implementada sobre un dispositivo de almacén secundario, en la que se pueden guardar valores de forma permanente para posteriormente recuperarlos y reprocesarlos.*

Desde el punto de vista del programador, los ficheros son objetos en los que se puede leer o escribir (o leer y escribir) datos. En un fichero existe siempre un *punto de lectura (o escritura)* que indica la posición del fichero en el que se va a leer (o escribir).

Dependiendo de cómo avanza dicho punto de lectura/escritura, se pueden clasificar los ficheros en ficheros *secuenciales* o de *acceso directo*. En los primeros, el punto de lectura/escritura avanza secuencialmente: tras la lectura o escritura de un dato avanza hasta quedar dispuesto para leer o escribir el siguiente dato. En un fichero de acceso directo, el punto de lectura/escritura se puede colocar en cualquier punto especificado por el programador.

Dependiendo de cómo se almacena la información, existen dos tipos de ficheros, los *ficheros de texto* y los *ficheros binarios*. En los ficheros de texto, los datos se almacenan como secuencias de carácteres (más concretamente, de sus códigos ASCII), mientras que en los ficheros binarios se suele almacenar la información tal y como se almacena en la memoria principal.

Por ejemplo, si se almacena un dato de tipo `int` en un fichero de texto, se almacenan los valores ASCII de los dígitos que lo forman. Si se almacena el mismo dato en un fichero binario, siempre ocupará 4 bytes (que es el tamaño de un dato de tipo `int` en memoria). Así, almacenar el valor

1232713 en un fichero de texto supone ocupar 7 bytes (tantos bytes como dígitos), mientras que almacenarlo en un fichero binario supone ocupar 4 bytes (se hará una copia del dato en el fichero tal y como se almacena en memoria, es decir, su representación binaria).

Los ficheros de texto pueden resultar más familiares en este punto de la asignatura. Por un lado, todos los programas que se han realizado en prácticas se almacenan en ficheros de texto, y se han creado y modificado utilizando editores de texto. Asimismo, en todos los programas diseñados hasta el momento se ha trabajado con la entrada y la salida estándar. Es posible ver a ambas como ficheros de texto: de alguna forma, la secuencia de caracteres que se teclean como un dato a leer por el programa por medio de la función `scanf()` equivale a la lectura de un dato de un fichero de texto. Algo similar ocurre con la función `printf()` y el hecho de que se visualicen los resultados en pantalla como una secuencia de caracteres.

Ambos tipos de ficheros presentan ventajas e inconvenientes. Una ventaja de los ficheros de texto es que su contenido, además de poder ser leído por cualquier editor de textos, es *portable* entre distintos computadores, ya que la información se almacena utilizando normalmente el código ASCII, como ya se ha comentado. Por contra en el manejo de los ficheros binarios pueden haber problemas de compatibilidad entre distintos computadores, ya que se pueden utilizar distintos modos de codificar los datos binarios.

Sin embargo, los ficheros binarios presentan la ventaja de tener un acceso a la información mucho más rápido que los ficheros de texto, ya que no hay que realizar ningún tipo de conversión, al realizarse una copia directamente de memoria a disco. Además, presentan la ventaja de que los datos tiene un tamaño uniforme; ello suele hacerlos más pequeños (como se vio en el ejemplo anterior sobre cómo almacenar en uno u otro tipo de fichero el valor 1232713) y además, que resulte más sencillo el acceso directo a sus datos: sabiendo el tipo de los datos almacenados, se sabe cuál es su tamaño, mientras que en un fichero de texto datos del mismo tipo pueden tener distinta longitud (el 10 y el 1000000, por ejemplo) lo que dificulta saber a priori dónde empieza y dónde acaba un dato particular.

Siempre que se trabaja con uno o varios ficheros hay que seguir estos pasos:

- Abrir el fichero.
- Trabajar con él (leer y/o escribir).
- Cerrar el fichero.

La apertura posibilita el tener una referencia válida a fichero y además permite indicar cuál va a ser su uso. Y el cierre permite garantizar que los datos se almacenen fisícamente de forma correcta, sin riesgo de corrupción o pérdida de datos.

7.2. Manipulación básica de ficheros en C.

7.2.1. La tupla FILE.

La biblioteca estándar de C incluye distintas estructuras de datos, funciones y procedimientos para trabajar con ficheros. Para poder utilizar estas herramientas hay que incluir el fichero de cabecera stdio.h.

Una de las principales estructuras de datos es una tupla, denominada `FILE`, que contiene en sus campos información sobre un fichero abierto: su estado, sus buffers, ...

En el lenguaje `C` un fichero siempre tiene asociada una tupla de tipo `FILE`, a la cual se accede por medio de un puntero de tipo base `FILE`. Por cada fichero que se quiera utilizar, hay que definir un puntero distinto; es lo que se conoce como el *descriptor* del fichero y se utiliza en la apertura y el cierre del fichero, además de para escribir o leer datos.

La forma general para esta declaración es,

FILE *<nombreDescriptor>;

Por ejemplo,

```
FILE *fdesc;
```

define un descriptor de fichero denominado `fdesc`.

7.2.2. Apertura y cierre de ficheros.

Como se acaba de comentar, un fichero siempre tiene asociado un descriptor que se ha de definir junto con el resto de variables de la función o procedimiento. Al igual que pasa con cualquier otra variable, al realizar la declaración no se le asigna ningún valor: se ha declarado un descriptor a fichero, pero dicho descriptor no tiene aún ningún fichero asociado. La operación de apertura hace justamente eso, asociar un fichero físico en disco (o en cualquier otro medio de almacenamiento secundario) a un descriptor de fichero.

La sintaxis para la apertura de un fichero responde a la siguiente forma general,

<nombreDescriptor> = fopen(<nombreFichero>, <modo>);

donde

- `nombreDescriptor`: es el puntero (previamente declarado) a una tupla `FILE`.
- `nombreFichero`: Es una cadena de texto que indica el nombre del fichero en disco. Se puede utilizar una valor de cadena constante (por ejemplo, `"datos.dat"`) o bien un identificador de una variable de tipo cadena, previamente declarada.
- `modo`: Es una cadena de texto que indica la forma en la que se va a abrir el fichero, cuál va a ser su uso y su tipo. Más adelante se comentarán los distintos modos (secciones 7.3 y 7.4), pero como ejemplo serviría la cadena `"wb"` que indica, como se verá, que se abre un fichero binario para escritura.

Cuando se realiza una llamada a `fopen` para abrir un fichero, la función devuelve un puntero a la estructura `FILE`. Si hay algún problema en la apertura (se quiere leer datos de un fichero que no existe, o no se tienen permisos para escribir en un fichero o se quiere escribir un nuevo fichero

y el disco está lleno, por ejemplo), el valor devuelto es el puntero nulo, `NULL`. Por lo tanto, es conveniente comprobar que el fichero se ha abierto correctamente antes de operar con él y, en caso contrario, tomar las acciones convenientes. Una secuencia típica para controlar esta circunstancia puede ser:

```
FILE *fptro;

fptro=fopen("mifichero","w");

if (fptro!=NULL) {
  /* (1) Código en caso de abrir el fichero */
}
else {
  /* (2) Código en caso de error */
}
```

Estas lineas de código abren el fichero `mifichero`; como se utiliza el modo `"w"`, se sabe que es para escritura y que se trata de un fichero de texto (al no utilizar el carácter `b` para indicar que es binario, se asume que se trata de un fichero de texto). A continuación, se comprueba si se ha podido abrir el fichero o no y, en función del resultado, se ejecutan unas instrucciones u otras.

Una vez que se procesan los datos y se termina de trabajar con el fichero *hay que cerrarlo* o, de lo contrario, no se puede garantizar un almacenamiento correcto de los datos. Para cerrar un fichero se ejecuta la función `fclose()`, pasándole como parámetro el descriptor del fichero a cerrar, es decir,

fclose(<nombreDescriptor>);

Según esto, el esquema mostrado en la secuencia del ejemplo anterior, quedaría completo de la siguiente forma:

```
FILE *fptro;

fptro=fopen("mifichero","w");

if (fptro!=NULL) {
  /* (1) Código en caso de abrir el fichero */

  fclose(fptro);
}
else {
  /* (2) Código en caso de error */
}
```

Final de fichero.

La biblioteca estándar incluye una función que indica si el punto de lectura/escritura se encuentra en el final de fichero o no. Su sintaxis es:

feof(<nombreDescriptor>);

y devuelve CIERTO si se ha alcanzado el final del fichero, FALSO en caso contrario.

Esta función es muy útil para leer datos de un fichero y detectar cuando se ha llegado al final. Un uso típico, en un esquema similar al de la secuencia anterior, sería el siguiente:

```
FILE *fptro;

fptro=fopen("mifichero","r"); /* r: lectura */

if (fptro!=NULL) {
  while (!feof(fptro)) {
     /* Código para procesar datos */
  }
  fclose(fptro);
}
else {
  /* Código en caso de error */
}
```

7.3. Ficheros de texto.

En un fichero de texto, la información se almacena como una sucesión de códigos ASCII. Los modos de trabajo válidos para este tipo de ficheros son:

- `r`, del inglés *read*, leer: Indica que el fichero se abre para lectura. El punto de lectura se sitúa al principio del fichero.
- `w`, del inglés *write*, escribir: Indica que el fichero se abre para escritura. El punto de escritura se sitúa al principio del fichero. Si el fichero físico no existe, se crea y si el fichero existe, se borra toda la información que había en él.
- `a`, del inglés *append*, añadir: Indica que el fichero se abre para escribir datos en él pero *al final*. Si el fichero físico no existe, se crea y si el fichero existe, el punto de escritura se sitúa al final del mismo sin borrar la información que había en él.
- `+`: Indica que el fichero se va a abrir para leer y escribir (`r+`) o para escribir y leer (`w+` o `a+`).

De estas opciones, una de las tres primeras es obligatoria y son mutuamente excluyentes. La última complementa a las anteriores.

7.3.1. Escritura en un fichero de texto

Para escribir datos en un fichero, se tiene que haber abierto en modo escritura. Se utiliza la instrucción `fprintf` cuya sintaxis es similar a la ya conocida `printf` con la diferencia que el

primer parámetro es el descriptor del fichero en el que se quiere escribir[1].

fprintf(<nombreDescriptor>,<cadenaDeControl>,<dato1>,<dato2>,...,<dato_k>);

Por ejemplo, el siguiente programa escribe en un fichero, `"vector.dat"`, los elementos de un vector que se leen de la entrada estándar:

```
int main() {
  int v[N];
  int  i;
  FILE *fptro;

  fptro=fopen("vector.dat","w");

  if(fptro==NULL)
     printf("\n\nERROR!!\n\n");
  else {
     for(i=0; i<N; i=i+1) {
        scanf("%d", &v[i]);
        fprintf(fptro, "%d", v[i]);
     }
     fclose(fptro);
  }
}
```

En este ejemplo, como cadena de control se ha utilizado la más básica, `" %d"`, pero la función `fprintf()` admite cualquiera, igual que `printf()`. Tal y como se ha utilizado, se escribirían los datos uno tras otro. Si se hubiera utilizado con otra cadena de control, por ejemplo,

```
fprintf(fptro," %d\n",v[i]);
```

su efecto hubiera sido el mismo que produciría en la pantalla: en cada línea, habría un elemento del vector. Otro ejemplo, la instrucción

```
fprintf(fp,"El valor es %f\n",val);
```

suponiendo que `val` contiene el valor `32.43`, escribe en el fichero asociado al descriptor `fp` la cadena "`El valor es 32.43`" y, después, cambia de línea. Esto es, se habría escrito en el fichero exactamente lo mismo que se hubiera visualizado por la pantalla. Esto es muy útil cuando se desea que un programa redireccione su salida de datos a un fichero, por ejemplo, si se desea estudiar tranquilamente que es lo que ocurre en una ejecución.

Además, al ser un fichero de texto, se puede visualizar el contenido del fichero y editarlo con cualquier editor de texto.

[1]Como ya se ha comentado, para el lenguaje `C` la entrada y salida estándar son como ficheros de texto; por lo tanto, el tratamiento es similar con la salvedad de que en los ficheros "no estándar" hay que indicar con qué fichero se va a trabajar.

7.3.2. Lectura en un fichero de texto

Para leer datos de un fichero de texto, se tiene que haber abierto en modo lectura `"r"`, y se utiliza la instrucción `fscanf`; como en el caso de la escritura, su sintaxis es similar a la de `scanf`, con la diferencia de que hay que indicar el descriptor del fichero del que se quiere leer.

fscanf(<nombreDescriptor>, <cadenaDeControl>, <dirDato1>, . . . , <dirDato_k>);

Por lo tanto,

```
fscanf(fp,"%d",&da);
```

lee un valor entero del fichero asociado a `fp` y almacena el dato leído en la variable `da`. El punto de lectura avanza al siguiente dato en el fichero. Nótese que, tal y como ocurre con `scanf`, se indica la dirección de la variable a leer, utilizando el operador `&`.

Como ejemplo de uso, se propone el siguiente programa, que lee los datos almacenados en el fichero `"vector.dat"` creado en el ejemplo de la subsección anterior, y los muestra por la salida estándar:

```
int main() {
  int v[N];
  int  i;
  FILE *fptro;

  fptro=fopen("vector.dat","r");

  if(fptro==NULL)
     printf("\n\nERROR!!\n\n");
  else {
     i=0;
     while (i<N && !feof(fptro)) {
        fscanf(fptro, "%d",&v[i]);
        printf("\nElemento v[%d]=%d,", i , v[i]);
        i=i+1;
     }

     fclose(fptro);
  }
}
```

7.4. Ficheros binarios.

Como ya se ha dicho, en un fichero binario la información se almacena tal y como se almacena en memoria; es decir, un dato de un tipo determinado ocupará en el fichero la misma cantidad de bytes que necesita para ser almacenado en memoria. Los modos de trabajo para los ficheros binarios son:

- `rb`, *read*, leer: Indica que se abre para lectura un fichero binario, situándose el punto de lectura al principio del fichero.
- `wb`, *write*, escribir: Indica que se abre un fichero binario para escritura. Como en el caso de los ficheros de texto, el punto de escritura se sitúa al principio del fichero. Si el fichero físico no existe, se crea y si el fichero existe, se sobreescribe.
- `ab`, *append*, añadir: Indica que un fichero binario se abre para añadir datos al final. Esto es, si el fichero existe el punto de escritura se sitúa al final; si no existe, se crea un fichero binario.
- `+`: Indica que el fichero se va a abrir para leer y escribir (`rb+`) o para escribir y leer (`wb+` o `ab+`).

De nuevo, una de las tres primeras opciones es obligatoria, y son mutuamente excluyentes entre sí, y la última complementa a las anteriores.

7.4.1. Escritura en un fichero binario.

Se utiliza la instrucción `fwrite` cuya sintaxis es:

fwrite(<dirDato>, <tamaño>, <elementos>, <nombreDescriptor>);

que escribe los datos que se encuentran en la posición de memoria indicada en `dirDato`. En la instrucción se indica el número de `elementos` a escribir, el `tamaño` de cada elemento y el descriptor de fichero asociado en el que se quiere escribir. El punto de escritura del fichero avanza tantos bytes como datos se han escrito.

Como ejemplo de uso, se vuelve a proponer el problema de leer los elementos de un vector de la entrada estándar, escribiéndolos en un fichero binario a medida que se leen:

```
1  int main() {
2    int v[N];
3    int i;
4    FILE *fptro;
5
6    fptro=fopen("vector.bin","wb");
7
8    if(fptro==NULL)
9       printf("\n\nERROR!!\n\n");
10   else {
11      for(i=0; i<N; i=i+1) {
12         scanf("%d", &v[i]);
13         fwrite(&v[i],sizeof(int),1,fptro);
14      }
15      fclose(fptro);
16   }
17 }
```

Si se analiza el uso que se ha realizado de la función `fwrite()` (línea 13),

```
fwrite(&v[i], sizeof(int), 1, fptro);
```

se ve que, de nuevo, el uso de la función `sizeof()` permite indicar de forma fácil para el programador el tamaño de un determinado tipo de dato. Asimismo, se utiliza el operador `&` para indicar la dirección del dato.

Pero la escritura puede expresarse de una forma más compacta. En el caso de los ficheros binarios se indica cuántos elementos se escribirán; por lo tanto, el ejemplo anterior podría haberse escrito de la siguiente forma:

```
int main() {
  int v[N];
  int i;
  FILE *fptro;

  for(i=0; i<N; i=i+1) {
    scanf("%d", &v[i]);
  }

  fptro=fopen("vector.bin","wb");

  if(fptro==NULL)
     printf("\n\nERROR!!\n\n");
  else {
     fwrite(v,sizeof(int),N,fptro);
     fclose(fptro);
  }
}
```

Sabiendo que `v` en el lenguaje `C` puede interpretarse como un puntero a la primera dirección de memoria de los elementos de dicho vector y que hay `N` elementos que almacenar, es posible escribir completamente el vector con una única llamada a `fwrite()`.

7.4.2. Lectura en un fichero binario.

Se utiliza en este caso la instrucción `fread`, cuya sintaxis es similar a la de la instrucción `fwrite`:

fread(<dirDato>, <tamaño>, <elementos>, <nombreDescriptor>);

En este caso, `dirDato` indica en qué posición de memoria se almacenará la información que se lea del fichero cuyo descriptor se indica en la llamada. De nuevo, se debe indicar el `tamaño` de cada elemento y el número de `elementos` a leer.

El punto de lectura del fichero avanza tantos bytes como datos se han leído. La función `fread` devuelve el número de elementos leídos, lo cual puede servir para comprobar si la operación se ha llevado a cabo con éxito o no (el valor devuelto coincide con `elementos`). El siguiente ejemplo muestra cómo usar esta instrucción:

```
1  int main() {
2    FILE *fptro;
3    int c, nb;
4
5    fptro=fopen("datos","rb");
6
7    if(fptro==NULL)
8       printf("\n\nERROR!!\n\n");
9    else {
10      while (!feof(fptro)) {
11        nb=fread(&c,sizeof(int),1,fptro);
12        if (nb==1)
13           printf("%d",c);
14      }
15      fclose(fptro);
16   }
17 }
```

El programa lee un fichero de enteros y muestra los datos leídos por pantalla. En primer lugar se abre el fichero (denominado `datos`) y se comprueba que se ha podido abrir. Entonces se inicia un bucle que finaliza cuando se alcanza el final de fichero. Después de leer cada dato se comprueba que el número de elementos leídos corresponde con los solicitados (uno en el ejemplo, línea 12) y en tal caso se muestra por la salida estándar.

Además de permitir comprobar que el número de elementos leídos es correcto, el hecho de tener que indicar en la llamada a `fread` el número de elementos a leer, permite la lectura de una estructura completa, tal y como ya se comentó al presentar la función `fwrite()`; por lo tanto, la siguiente llamada a `fread`,

```
fread(v, sizeof(int), N, fptro);
```

permite leer N enteros del fichero asociado a `fptro` y almacenarlos en el vector `v`.

7.4.3. Un ejemplo: escritura y lectura de una lista.

Como ejemplo de uso de cada tipo de fichero, y para comparar como se debe trabajar en con uno u otro tipo, se propone la escritura de dos procedimientos, uno que permita guardar los datos de una lista en un fichero y otro que permita recuperarlos y almacenarlos de nuevo en una lista.

Se asume que la definición de la lista responde a la siguiente definición de tipos:

```
/* Definición de datos en la lista */
typedef struct {
  char nombre[80];
  char telefono[15];
  int edad;
} tPersona;
```

```
/* Definición de la tupla básica */
struct nodo {
  tPersona dato;
  struct nodo *sig;
};

/* Definición del tipo nodo básico */
typedef struct nodo tNodo;

/* Definición de un tipo puntero al nodo básico */
typedef tNodo *tDirNodo;

/* Definición de la lista */
typedef struct {
  int longitud;
  tDirNodo primero, ultimo;
} tLista;
```

- `guardaListaTexto`: Procedimiento que almacena los datos de una lista de tipo base `tPersona` (nombre, teléfono y edad) en un fichero de texto:

```
1  void guardaListaTexto(tLista l, char *fichero) {
2  /* Pre: l es una lista ya creada y no vacía y */
3  /* fichero es una cadena con un identificador de */
4  /* fichero válido y se puede escribir en él */
5
6    tDirNodo lptro;
7    tPersona aux;
8    FILE *fptro;
9
10   fptro=fopen(fichero,"w");
11
12   if(fptro==NULL)
13      printf("\n\nERROR!!\n\n");
14   else {
15      lptro=primero(l);
16      while (lptro!=ultimo(l)) {
17        aux=dato(l,lptro);
18
19        fprintf(fptro,"%s\n",aux.nombre);
20        fprintf(fptro,"%s\n",aux.telefono)
21        fprintf(fptro,"%d\n",aux.edad);
22
23        lptro=siguiente(l,ptro);
24      }
25      aux=dato(l, lptro);
26
27      fprintf(fptro,"%s\n",aux.nombre);
28      fprintf(fptro,"%s\n",aux.telefono)
29      fprintf(fptro,"%d\n",aux.edad);
30
31      fclose(fptro);
32   }
```

```
33
34 /* Post: el fichero contiene los datos de personas */
35 /* incluidas en la lista l, de forma que hay un dato */
36 /* por línea */
37 }
```

De esta forma se almacenan los datos en un fichero de texto en el que cada línea almacena un dato (tres líneas almacenan los datos de una persona).

- `recuperaListaTexto`: Si los datos están almacenados en un fichero de texto, tal y como se ha hecho en el apartado anterior (y es necesario saber que hay un dato por línea), el código para leerlos y almacenarlos en una lista es:

```
1  void recuperaListaTexto(tLista *l, char *fichero) {
2  /* Pre: fichero es una cadena que identifica a un */
3  /* fichero en el que se ha almacenado datos de una */
4  /* persona, un dato por línea */
5
6    tPersona pa;
7    FILE *fptro;
8
9    *l=crearLista();
10
11   fptro=fopen(fichero,"r");
12
13   if(fptro==NULL)
14      printf("\n\nERROR!!\n\n");
15   else {
16      while (!feof(fptro)) {
17       /* "%[^\n]": leer una cadena de caracteres, que */
18       /* puede incluir espacios, hasta que se lea \n */
19
20         fscanf(fptro," %[^\n]",pa.nombre);
21         fscanf(fptro," %[^\n]",pa.telefono);
22         fscanf(fptro,"%d",&(pa.edad));
23         if (!feof(fptro)) {
24               almacenar(l,pa);
25          }
26      }
27      fclose(fptro);
28   }
29
30 /* Post: la lista l está formada por los datos de */
31 /* persona almacenados inicialmente en fichero */
32 }
```

- `guardaListaBin`: Procedimiento para escribir los datos almacenados en una lista, como la definida anteriormente, en un fichero binario.

```
1  void guardaListaBin(tLista l, char *fichero) {
2  /* Pre: l es una lista ya creada y no vacía y */
3  /* fichero es una cadena con un identificador de */
```

```
/* fichero válido y se puede escribir en él */

  tDirNodo lptro;
  tPersona aux;
  FILE *fptro;

  fptro=fopen(fichero,"wb");

  if(fptro==NULL)
     printf("\n\nERROR!!\n\n");
  else {
     lptro=primero(l);
     while (lptro!=ultimo(l)) {
       aux=dato(l,lptro);
       fwrite(&aux,sizeof(tPersona),1,fptro);
       lptro=siguiente(l,lptro);
     }
     aux=dato(l,lptro);
     fwrite(&aux,sizeof(tPersona),1,fptro);

     fclose(fptro);
  }
/* Post: el fichero contiene los datos de personas */
/* incluidas en la lista l, almacenados con formato */
/* binario */
}

```

- `recuperaListaBin`: Los datos están en un fichero binario; por lo tanto, se utilizará `fread` para leerlos. La función lee posiciones consecutivas del fichero, y no tiene forma de distinguir si se trata de una tupla, un vector, un entero, ... El procedimiento debe saber cómo se almacenaron los datos para leerlos en el mismo orden.

```
void recuperaListaBin(tLista *l, char *fichero) {
/* Pre: fichero es una cadena que identifica a un */
/* fichero en el que se ha almacenado datos de una */
/* persona, siendo cada dato de tipo tPersona */

  tPersona pa;
  FILE *fptro;
  int a;

  *l=crearLista();
  fptro=fopen(fichero,"rb");

  if(fptro==NULL)
     printf("\n\nERROR!!\n\n");
  else {
     while (!feof(fptro)) {
       a=fread(&pa,sizeof(tPersona),1,fptro);
       if (a>0)
         almacenar(l,pa);
     }
     fclose(fptro);
```

```
22    }
23  /* Post: la lista l está formada por los datos de */
24  /* persona almacenados inicialmente en fichero */
25  }
```

7.5. Ficheros de acceso directo.

Hasta este punto, se ha trabajado con los ficheros en modo secuencial, es decir, se abre el fichero y se va leyendo (o escribiendo) uno o varios datos *uno detrás del otro*.

La biblioteca estándar de `C` dispone de órdenes que permiten modificar el punto de lectura o escritura y posicionarlo en cualquier punto del fichero. De esta forma se puede acceder directamente a un dato, sin necesidad de pasar por todos los que le preceden.

Los ficheros que permiten trabajar de este modo se conocen como ficheros de *acceso directo* o aleatorios. El lenguaje `C` permite acceder de forma directa a cualquier fichero, si bien suele ser más común trabajar de esta forma con ficheros binarios, en los que el espacio necesario para almacenar los datos es uniforme.

Asimismo, el acceso directo puede ser un buen complemento para la posibilidad de abrir un fichero para lectura y escritura al mismo tiempo: cuando el acceso es secuencial no es una opción muy interesante, pero si se puede acceder directamente a un punto determinado del fichero entonces sí tiene más utilidad (sobreescribir una dato determinado después de leerlo y verificar que cumple una condición, por ejemplo).

La orden que permite mover el punto de lectura/escritura dentro de un fichero es

fseek(<nombreDescriptor>, <desplazamiento>, <posición>)

donde

- `nombreDescriptor`: es el descriptor del fichero.
- `desplazamiento`: es el número de bytes que se quiere desplazar el punto de lectura/escritura, que puede ser cero, positivo o negativo.
- `posición`: indica el punto del fichero respecto al que se aplica el desplazamiento. Puede tener los siguientes valores:
  - SEEK_SET, desde el principio; el desplazamiento puede ser cero o positivo.
  - SEEK_CUR, desde la posición actual del punto de lectura o escritura.
  - SEEK_END, desde el final; el desplazamiento, en este caso, debe ser cero o negativo.

Por ejemplo, si se tienen en un fichero datos del tipo `tPersona`, definido como en la subsección anterior, y se ha declarado el descriptor `fptro` para acceder al fichero, la ejecución de la instrucción

```
fseek(fptro, 10*sizeof(tPersona), SEEK_SET);
```

posiciona el punto de lectura o escritura al final de la décima persona.

La ejecución de

```
fseek(fptro, 1, SEEK_CUR);
```

desplaza un byte el punto de lectura o escritura respecto a su posición actual. Si lo que se desea es un desplazamiento hasta los datos de la persona anterior, se puede utilizar lo siguiente,

```
fseek(fptro, -sizeof(tPersona), SEEK_CUR);
```

Y, la ejecución de

```
fseek(fptro, -10*sizeof(tPersona), SEEK_END);
```

posiciona el punto de lectura o escritura 10 personas antes del final del fichero. O, para colocar el punto de lectura/escritura al comienzo de la edad de la última persona en el fichero,

```
fseek(fptro, -sizeof(int), SEEK_END);
```

ya que la edad es el último campo y es de tipo entero.

7.6. Resumen y Consideraciones de Estilo.

Los ficheros proporcionan al programador una herramienta para almacenar sus datos de forma no volátil para poder recuperarlos y procesarlos. En este tema se han presentado dos tipos de ficheros que se diferencian en la forma en que almacenan la información; cada uno presenta ventajas e inconvenientes cuando se comparan, pero es preciso recordar que ambos precisan de un tratamiento similar en la necesidad de abrirlos antes de trabajar y después *cerrarlos* convenientemente para asegurar que los datos han sido convenientemente almacenados.

Igual que cuando se habla de cómo nombrar otros objetos, es conveniente elegir nombres significativos para los ficheros que se creen. Asimismo, es muy importante tener en cuenta cómo se han almacenado los datos a la hora de recuperarlos.

Evidentemente, lo visto en este tema es sólo el principio de la gestión de ficheros. De hecho, cuando se pretende manejar un gran volumen de datos se recurre a las *bases de datos* y a los programas y lenguajes orientados al manejo y al diseño de bases de datos y de sistemas de bases de datos. Estos ofrecen al usuario diversas herramientas para realizar la gestión de los datos: almacenar la información y recuperarla, leerla y modificarla son sólo algunas de las posibilidades.

7.7. Bibliografía.

1. “The C Programming Language”. Brian W. Kernigham & Dennis M. Ritchie. Prentice-Hall Inc. 1988.

7.8. Problemas Propuestos.

1. Se ha definido correctamente la lista `l` y se ha abierto correctamente un fichero binario para escritura sobre `fptro`. ¿Por qué se considera incorrecta la siguiente escritura de datos de una lista?

   ```
   fwrite(l,sizeof(tNodo),longitud(l),fptro);
   ```

   Nota: `longitud(l)` se supone que es una función correcta que devuelve la longitud de la lista `l`.

2. ¿Y esta?

   ```
   aux=l;
   while (aux!=NULL) {
       fwrite(aux, sizeof(tNodo), 1, fptro);
       aux=aux->sig;
   }
   ```

3. Escribir un programa que lea datos enteros de un fichero de texto; si el valor leído es distinto de cero se escribirá en otro fichero de texto, pero si el valor es nulo, se debe escribir la suma acumulada de todos los elementos anteriores.

   Es decir, si la secuencia de datos que se va leyendo es

   1, 45, 6, 0, 10, 123, 0, 2, ...

   la secuencia de datos a escribir será

   1, 45, 6, 52, 10, 123, 237, 2, ...

4. Se dispone de dos ficheros de texto, `dnis.dat` y `notas.dat`, que contienen, respectivamente, DNIs y notas de 100 alumnos. Los datos guardados en cada fichero son correlativos; es decir, la primera nota en el fichero `notas.dat` es la correspondiente al alumno cuyo DNI es el primero en `dnis.dat`, la segunda nota corresponde al segundo DNI y así, sucesivamente. Hay tantos DNIs como notas.

   a) Este almacén de datos no nos acaba de convencer, porque pone en peligro el mantenimiento de la coherencia de información. Se quiere guardar la información en un vector.

   Definir la estructura de datos más adecuada para almacenar de forma coherente la información, es decir, definir el tipo base más adecuado y el propio vector.

   Escribir un procedimiento que lea los datos de los ficheros y los almacene en un vector, de acuerdo al tipo definido.

   b) Dado un vector como el obtenido en el apartado anterior, se asume que la nota de cada alumno se refiere a la obtenida en el primer parcial y ahora se quiere calcular la nota total.

   Escribir un procedimiento que sobreescriba en el vector la calificación del 1^{er} parcial con la total de la asignatura, sabiendo que:

   - por cada alumno, se lee por teclado su nota del examen final, y su nota de prácticas que, sumadas, permiten calcular la nota final.
   - si el resultado es mayor o igual que 4, la calificación se obtiene mediante la expresión

     $$\text{máx}((EP \times 0{,}35 + NF \times 0{,}65), NF)\ ;$$

- si no, la calificación coincide con la nota final.

Además, el procedimiento devolverá la **nota media de todos** los alumnos y la **nota media de los alumnos aprobados**.

5. Se dispone de un fichero de texto con los datos sobre la pluviosidad a lo largo de los 365 días de un año. Sabiendo que el primer día del año es lunes y que el año no es bisiesto, se pide:

   a) Realizar una función o procedimiento que lea los datos del fichero y los almacene en una matriz con 53 filas y 7 columnas (cada fila representa una semana y cada columna un día de la semana).

   b) Realizar una función o procedimiento que dada una matriz como la del apartado anterior calcule la media pluviométrica, por semana, de las primeras 52 semanas del año.

   En ambos casos se debe definir la **precondición** y la **postcondición**.

6. En un fichero de texto se ha almacenado una matriz de reales, de tamaño N $\times$ N, triangular superior. No están almacenados los ceros.

   Hay que escribir un programa que la imprima por pantalla, con los ceros en su sitio.

7. En un fichero de texto, se ha almacenado una frase (en letras mayúsculas) encriptada de acuerdo al siguiente algoritmo: independientemente de dónde estén originalmente las letras (o los espacios en blanco), primero se han agrupado los carácteres de cinco en cinco y, después, se ha aplicado el algoritmo de Julio César: cada carácter se desplaza una cantidad fija. Esa cantidad se puede conocer, ya que el primer carácter del fichero será el carácter obtenido al desplazar la segunda letra. Es decir, el primer carácter aparece repetido, codificado y sin codificar, lo que permite calcular el desplazamiento aplicado en la codificación (según esto, tened en cuenta que el texto comienza de verdad a partir del segundo carácter del fichero).

   Escribir un programa que permita visualizar el texto desencriptado.

8. Una librería almacena información sobre sus 100 mejores clientes. Cada vez que uno de estos clientes hace una compra, se almacena en un fichero el código del cliente (un valor entre 0 y 99), el precio en euros del libro comprado y el número de unidades compradas de dicho libro. Toda esa información se almacena en una misma línea.

   La libreria premia con puntos a estos clientes en función de las compras realizadas: 3 puntos por libro comprado y 2 puntos por cada 10 euros en compras.

   Se pide:

   a) Sabiendo que para procesar la información almacenada en el fichero, se quiere recuperar ésta en un vector de tamaño 100, indicar cuál sería la definición del vector y de su tipo base, teniendo en cuenta la información que es preciso almacenar por cada cliente.

   b) Definir una función o procedimiento que dado un fichero de las características indicadas, recupere la información sobre un vector como el definido en el apartado anterior y devuelva las cantidades gastadas por cada cliente y el número de puntos que le corresponde.

9. Para cada uno de los alumnos presentados al examen final de una determinada asignatura se ha almacenado su DNI, su nota en el examen y su nombre (los tres datos en la misma línea) en un fichero de texto.

   a) Definir una función o procedimiento que dado un fichero de las características indicadas, calcule la nota media de los alumnos presentados al examen, la nota media de los alumnos suspensos y la nota media de los alumnos aprobados.

b) Definir una función o procedimiento que a partir del fichero y de las notas medias de aprobados y suspensos, produzca una lista enlazada en la que se almacenará el DNI y el nombre de cada alumno cuya calificación sea superior a la media de los suspensos e inferior a la media de los aprobados.

Nota: La respuesta debe incluir la definición del tipo base de la lista y la redefinición del tipo `tLista` con respecto al visto en teoría. Se pueden utilizar las operaciones de manejo de listas estudiadas en teoría.

¿Cómo encadenar las llamadas a ambos procedimientos o funciones para realizar correctamente el proceso completo (es decir, que a partir del fichero se obtenga la lista enlazada)?

10. En un fichero de texto se ha almacenado el padrón municipal. Por cada habitante se almacena en una línea

```
<nombre> <dni> <fechaNacimiento> <direccion>
```

Se asume que `nombre`, `dni` y `direccion` son cadenas (sin blancos) que representan, respectivamente, el nombre, el DNI y la dirección postal de cada habitante y que `fechaNacimiento` es una tupla con tres campos enteros, uno para el día, otro para el mes y otro para el año.

Hay que escribir una función o procedimiento que dado un fichero de las características indicadas, escriba en otro fichero de texto los nombres y DNIs de los habitantes con derecho a voto en las elecciones del 13 de Junio de 2004.

11. Una compañía de seguros necesita realizar una serie de estudios estadísticos sobre la repercusión de la huelga de gruistas. Para dicho estudio dispone en un fichero de texto de la siguiente información por línea,

```
<matricula>  <horas>  <robo>
```

en la que `matricula` es una cadena de 9 caracteres para almacenar la matrícula de un vehículo, `horas` es un valor real que indica el número de horas que ha permanecido un vehículo sin ser recogido y `robo` es un `boole` que indica si dicho vehículo ha sufrido o no un robo antes de ser recogido.

Hay que escribir una función o procedimiento que dado un fichero de las características indicadas, obtenga un nuevo fichero en el que se escribirán las matrículas de los coches que han sufrido un robo, además de devolver la media de horas de abandono de los vehículos que han sufrido un robo y la media de horas de abandono de los vehículos que no han sufrido un robo.

12. En un almacén de productos de papelería se ha observado que hay productos antiguos que conviene liquidar. Para ello se establece que se rebajarán a la mitad (50 %) los productos que lleven más de 24 meses en el almacén y que el descuento será del 35 % para los que lleven entre 8 y 24 meses.

En un fichero de texto se ha almacenado la siguiente información por línea:

```
<referencia> <meses> <precio>
```

en la que `referencia` es el código del producto (una cadena sin blancos), `meses` es la cantidad de meses que lleva el producto en el almacén y `precio` es su precio de venta al público.

Hay que escribir una función o procedimiento que dado un fichero de las características indicadas, escriba en otro fichero de texto la siguiente información relativa a los productos: por línea,

```
<referencia> <descuento> <nuevoPrecio>
```

en la que `referencia` representa la misma información que antes, `descuento` es el descuento aplicado y `nuevoPrecio` el nuevo precio venta al público.

13. El departamento de finanzas de una empresa recibe peticiones de compra de material y las va almacenando en un fichero de la siguiente forma:

    - código (un entero)
    - descripción (una cadena)
    - precio (un float)
    - cantidad (un entero)

    Es decir, cada ítem de información se guarda en una nueva línea.

    Cada semana se procesan esas peticiones, de forma que las que suponen un gasto de hasta 50 euros se autorizan automáticamente, pero las que superan esa cantidad se escriben en otro fichero a la espera de que el Director dé el visto bueno a ese gasto.

    Hay que escribir una función o un procedimiento (justificar la elección) que, dado un fichero como el anterior, calcule el total de gastos autorizados automáticamente y, además, escriba en un fichero las peticiones que se deben pasar a dirección, manteniendo el mismo formato de datos.

14. Una empresa de productos lácteos tiene en marcha una promoción: entre todos los clientes que envíen 100 tapas de sus productos sorteará 10 ordenadores portátiles.

    Para almacenar la información de todos los clientes que toman parte en la promoción se dispone de un fichero de texto en el que se guarda la siguiente información por cada cliente:

    ```
    <nombre>\n
    <direccion>\n
    <cantTapas>\n
    ```

    en la que `nombre` y `direccion` son cadenas, que pueden contener blancos, y `cantTapas` es un entero que indica cuántas tapas ha enviado un cliente. Cada ítem de información se almacena en una línea.

    a) Hay que escribir una función o procedimiento que dado un identificador de fichero (que será de las características indicadas), permita introducir nuevos datos: se leerá de teclado la información relativa a un nuevo cliente (nombre, dirección y cantidad de tapas) y se añadirá dicha información al final del fichero.

    b) Hay que escribir una función o procedimiento que dado un identificador de fichero, como el descrito anteriormente, obtenga como resultado otro nuevo fichero en el que sólo figurarán aquellos clientes que realmente hayan enviado 100 o más tapas. El nuevo fichero tendrá la misma estructura que el original,

    ```
    <nombre>\n
    <direccion>\n
    <cantTapas>\n
    ```

    c) Para realizar el proceso del sorteo se manejará una lista enlazada cuyos nodos contendrán la misma información básica, `nombre`, `direccion` y `cantTapas`, por cada cliente. Hay que realizar la redefinición del TAD lista estudiado en teoría para que permita almacenar dicha información, adaptando las estructuras de datos e indicando qué operaciones quedan afectadas y cómo.

    d) Se supone que se dispone de la función `sorteo`, cuyo prototipo es el siguiente:

    ```
    int sorteo (int total);
    /* Pre: total es un valor entero mayor que 1 */
    /* Post: devuelve un valor aleatorio entre 1 y total */
    ```

Hay que escribir una función o procedimiento que dado el identificador del fichero de clientes, una lista de clientes y un valor entero que indica cuántos clientes hay en el fichero, obtenga un valor aleatorio entre 1 y dicho valor realizando una llamada a `sorteo` y añada al final de la lista los datos del cliente cuya posición en el fichero coincida con el resultado de la llamada.

e) Hay que escribir una función o procedimiento que implemente el sorteo de los 10 ordenadores portátiles: dado el identificador del fichero de clientes, hay que determinar cuántos clientes participan en el sorteo y crear una lista en la que figuren los 10 clientes premiados. Para ello hay que utilizar llamadas al código desarrollado en el apartado anterior. Además, se debe asegurar que en cada hogar sólo se reparte un permio (es decir, en la lista de premiados no hay direcciones repetidas).

15. Una empresa de trasporte urbano tiene asignado cada autobús disponible a una línea de servicio. Esta información se almacena en un fichero, de modo que para cada autobús se guardan 5 líneas con la siguiente información (cada ítem en una línea):

    - Matrícula (una cadena)
    - Código numérico (un entero: `1`,`2`,`21`,`3`,...)
    - Nombre de línea (una cadena: `L10`, `L12`, `L12-BIS`, `L3`,...)
    - Fecha de asignación a esta línea
    - Fecha de antigüedad del autobús

    Ambas fechas constan de 3 números enteros (día, mes y año), por ejemplo: "`15 09 1988`". Hay que escribir una función o procedimiento (justificar la elección) que, *dado un fichero como el anterior, y dado el código de una línea de autobuses*, **escriba en un fichero las matrículas de todos los autobuses asignados a dicha línea, así como su fecha de antigüedad**.

    También debe calcular **cuál es la matrícula y fecha de antigüedad del más antiguo de todos los autobuses** registrados en el fichero y **devolver esta información** como resultado.