Archivos

Hasta el momento todo nuestro trabajo de programación, nuestro estudio con relación al tema en las distintas materias de la carrera, estuvo basado en la lectura y escritura de estructuras de datos almacenadas en la memoria principal. Pasamos de manejar sencillos programas con unas pocas variables simples a otros muchos más interesantes y complejos donde combinamos distintas estructuras de programación con funciones y estructuras de datos también más complejas, pero aún así, nuestros programas tenían por objeto más aprender a programar que generar sistemas que puedan prestar servicio en la vida real. Las limitaciones de los programas desarrollados a la fecha tienen que ver con la necesidad de almacenar la información que éstos generan o utilizan de manera tal que esté disponible sin necesidad de cargarla en la memoria cada vez que el programa se inicia, ni definir su tamaño previamente: y para eso están los archivos.

Prácticamente toda la actividad que desarrollamos en una computadora está basada en información almacenada en archivos: cuando arrancamos la máquina el sistema operativo lee y ejecuta un conjunto de archivos que permiten que ésta pueda ser utilizada; cuando iniciamos un programa cargamos en la memoria por medio del sistema operativo un archivo específico; cuando desde un programa en particular generamos un documento estamos creando un archivo; cuando escribimos un programa creamos un archivo de texto con instrucciones en C; cuando compilamos un programa fuente estamos creando un archivo ejecutable a partir de lo que escribimos más enlaces a otros archivos existentes, etc., etc.. Veamos entonces una definición de archivo.

> **Un archivo puede definirse como un conjunto de información relacionada, almacenada en un dispositivo externo –generalmente un disco u otro dispositivo de almacenamiento masivo-, que tiene un nombre para su identificación, y una ubicación –ruta- dentro del dispositivo**.

También podríamos definirlo como

> **Una estructura de datos almacenada externamente compuesta por un conjunto de elementos relacionados de longitud variable (recuérdese que las estructuras de datos residentes en memoria principal tienen un tamaño fijo establecido en su definición).**

La longitud de la estructura de datos archivo tiene como límite la capacidad de la unidad donde reside, que en general es mucho más grande que la memoria RAM. La posibilidad de almacenar grandes volúmenes de información en archivos permite independizar los datos de los programas que usan estos datos.

Clasificación de los archivos

Una primera clasificación posible de los archivos sería la determinada por el uso que el usuario o el sistema operativo hacen de ellos. Así tenemos:

- **Archivos de sistema**: son aquellos utilizados por el sistema operativo para todas las funciones de administración de la máquina. En general no están disponibles para el usuario, salvo en operaciones de configuración avanzada.

- **Archivos ejecutables**: son los programas o aplicaciones que permiten a los usuarios realizar tareas concretas, tales como un documento de texto, un gráfico, u otro programa. La única acción posible sobre este tipo de archivos es su ejecución y uso.

- **Archivos de datos:** son los archivos que el operador crea a partir de una aplicación, como un procesador de texto, un editor de imágenes, una planilla de cálculo, etc. Cada una de estas aplicaciones permite la manipulación de archivos específicos (creación, modificación, asignación de nombre y ruta, etc.). En cualquier caso, los programadores de estas aplicaciones determinaron el formato interno de sus archivos de manera tal que los usuarios vean sus archivos tal como lo crearon, sin necesidad de saber cómo es la organización interna de los bytes que lo componen. Por ejemplo, en un archivo de texto plano se almacenan sólo los caracteres escritos por el usuario (letras, números, espacios y fin de línea); en un archivo de texto con formato, además de los caracteres es necesario almacenar información sobre el formato: tipo de letra, párrafo, colores, etc. Al crear y abrir el archivo con una misma aplicación –u otra compatible capaz de interpretar la organización interna de esos archivos- la información incorporada se reproduce tal como fue creada.

Cada sistema operativo reconoce un conjunto propio de archivos, esto es, la forma de identificarlos, de organizarlos, y de asociarlos con acciones específicas. La asociación se establece por medio de un identificador que forma parte del nombre del archivo: la extensión. Por ejemplo, en Windows, si un archivo tiene la extensión .exe, el sistema operativo lo identifica como un archivo ejecutable, y por lo tanto intentará cargarlo en la memoria cuando se solicite su ejecución; si es un archivo válido, se abrirá un programa de aplicación. Además, al instalar una aplicación, se notifica al sistema operativo que relacione los archivos con una determinada extensión con esa aplicación. De esta manera al solicitar la apertura de un archivo –por ejemplo haciendo doble click sobre él- el sistema operativo abrirá el programa relacionado con la extensión del archivo, y luego intentará abrir ese archivo dentro de la aplicación.

Un tipo particular de archivo de datos es aquel que es generado desde una aplicación que desea almacenar elementos de información sobre alguna “cosa” en particular. Supongamos que esa “cosa” sea un conjunto de artículos que un comercio vende, y que identifica de la siguiente manera:

- Código de artículo (char[5])
- Descripción (char[30])
- Precio unitario (float)
- Stock (entero)

Cada artículo está definido por un valor de código (que se diferencia de todos los demás códigos), una descripción, un precio unitario y un stock. Como ya hemos visto esta forma de agrupar información se denomina **registro**; también sabemos **que un registro está compuesto por un conjunto de campos**. Luego si necesitamos almacenar información de un

conjunto de artículos, y cada uno de esos artículos necesita de un registro, **nuestro archivo estará compuesto por un conjunto de registros de igual tipo.**

Todas las organizaciones necesitan almacenar información sobre “cosas”, entidades, que son de su interés. Una empresa necesitará guardar información sobre empleados, productos, proveedores, ventas, etc.; una Universidad sobre alumnos, docentes, materias, carreras, etc.

Por lo general, los sistemas que gestionan las actividades de esas organizaciones tienen un conjunto de archivos (uno por cada una de las entidades) **relacionados, organizados en una base de datos.**

Podríamos entonces establecer la siguiente jerarquía en la organización de la información:

Conjunto de bits: carácter
Conjunto de caracteres: campo
Conjunto de campos relacionados: registro
Conjunto de registros relacionados: archivo
Conjunto de archivos relacionados: Base de Datos.

Volviendo a los sistemas de gestión, es común que las instituciones necesiten almacenar cientos, miles o cientos de miles de registros en sus archivos. Se comprenderá entonces que es de suma importancia que en la organización de estos archivos se prevean mecanismos que faciliten el acceso a cada uno de los registros, y que garanticen que la información almacenada sea consistente, confiable; de otro modo los sistemas serían demasiado lentos y la información que proporcionen sería dudosa. Esto será el objeto de estudio de otras materias; para nuestro caso nos limitaremos a definir las características de los archivos que forman parte de esas bases de datos.

Luego de diversas experiencias y estudios, se llegó a la conclusión que la mejor forma de organizar información en los archivos era utilizar registros de longitud fija, que todos los registros de un archivo sean de un mismo tipo, y que cada uno de esos registros tenga un campo clave, que no repita su valor en el resto del archivo. Para el ejemplo que dimos, el campo clave sería el código de artículo (identifica a cada artículo en particular y no se repite).

Al conocer la longitud de cada registro, se puede acceder fácilmente al resto de los registros, desplazándose una cantidad de bytes hacia adelante o hacia atrás en el archivo. Estos archivos pueden verse como tablas, tal como se muestra más abajo para nuestro ejemplo:

Código de artículo	Descripción	Precio Unitario	Stock
“aaaa”	“Zanahoria”	5.3	150
“bbbb”	“Uva”	9.60	200

Este tipo de archivos es el que utilizaremos en nuestros programas. Para definir cada uno de los campos, y por lo tanto el contenido y tamaño de cada registro, utilizaremos como molde las clases que sean necesarias. Las propiedades definidas dentro de cada clase serán los campos de nuestros archivos.

Archivos en C

En C cada archivo es visto como con un flujo secuencial de bytes. Al establecer la conexión con el archivo físico en la apertura lo asocia con un flujo, o corriente, que puede ser visto como un dispositivo lógico: no será necesario hacer nuevamente referencia al archivo físico, sino que todas las acciones se realizarán por medio de este flujo. **Los flujos proporcionan un canal de comunicación entre los archivos y los programas.**

Al abrir un archivo, la función **fopen(),** que se verá más adelante, devuelve un puntero a una estructura **FILE**; esta estructura contiene toda la información necesaria para procesar el archivo. Cada archivo que se abra tendrá su propio **puntero FILE.** Luego de la apertura, cada vez que se quiera operar sobre un archivo se lo hará por medio del **puntero FILE.**

C proporciona un conjunto de funciones para realizar operaciones con archivos. No dispone de formatos preestablecidos, ni funciones para controlar que lo que se escriba o lea sea correcto, por lo que es tarea del programador garantizar lo anterior.

Operaciones básicas sobre archivos

Apertura de un archivo

Para comenzar a trabajar con el archivo hay que establecer la conexión entre el programa y el archivo, o sea realizar su apertura. La función encargada es fopen() (del inglés f –file, archivo-, open-abrir-). Su prototipo es:

FILE * fopen(“ruta_nombrearchivo”, “modo_de_apertura”)

La función devuelve un puntero FILE en caso de que la operación haya sido exitosa; si la apertura no pudo hacerse devuelve la **constante NULL.**

El tipo de datos FILE es una estructura definida en las librerías de C (stdio.h o cstdio si usamos C++). Contiene toda la información necesaria para vincular el archivo con el programa, y operar sobre él. Desde el punto de vista práctico no es necesario conocer los detalles de la estructura FILE. Baste señalar que contiene un descriptor de archivo que se vincula con el componente del sistema operativo que se encarga de administrar ese archivo en particular. Como veremos más adelante, todas las funciones de archivo usan el puntero FILE.

Como parámetros **fopen()** tiene dos cadenas: una indica la ruta y el nombre del archivo que se quiere abrir, y la otra el modo en que se quiere abrir el archivo.

El modo de apertura determina lo que se puede hacer con el archivo. Los modos posibles son:

- **"r":** Apertura para lectura de un archivo de texto (read). El archivo debe existir.
- **"w":** Apertura para la escritura de un archivo de texto (write). Si el archivo no existe lo crea; si existe lo reemplaza por un archivo vacío. Todos los registros –si los tuviera- se pierden.
- **"a":** Apertura de un archivo de texto para agregar registros (append). Si no existe lo crea.
- **"rb":** Apertura para lectura de un archivo binario (read). El archivo debe existir.
- **"wb":** Apertura para la escritura de un archivo binario (write). Si el archivo no existe lo crea; si existe lo reemplaza por un archivo vacío. Todos los registros –si los tuviera- se pierden.
- **"ab":** Apertura de un archivo binario para agregar registros (append). Si no existe lo crea.

Si a cualquiera de los modos se le agrega el signo + (por ejemplo "rb+"), se le agrega la funcionalidad que el modo no tiene (con "rb+" se abre un archivo binario para lectura y escritura).

Por ejemplo, si se quisiera abrir el archivo articulo.dat, que se encuentra en la raíz del disco C:, para leer los registros que el archivo tiene:

```
FILE *pArticulo;

pArticulo = fopen("c :\\articulo.dat", "rb");

o

pArticulo = fopen("c : / articulo.dat", "rb");
```

Luego de la llamada a la función, debe comprobarse si se pudo hacer la apertura. Como se dijo, si la función no pudo hacer la apertura el puntero FILE tendrá como valor NULL.

```
if (pArticulo == NULL)
{
    cout <<"No pudo abrirse el archivo";
    exit(1);
}
```

En este caso se decidió terminar el programa, ya que sin el archivo no puede continuar su ejecución.

Como se dijo, el modo de apertura determinará las operaciones que se pueden hacer con el archivo. En este caso, por ejemplo, no se podría escribir sobre el archivo porque la apertura se hizo para lectura.

Cierre de un archivo

Para cerrar un archivo, esto es, clausurar la vinculación entre el programa y el archivo, se utiliza la función **fclose().** El único parámetro que tiene la función es el puntero FILE.
Por ejemplo, para cerrar el archivo articulo.dat

fclose(pArticulo);

ya que pArticulo se utilizó para la apertura del archivo.

Siempre debe cerrarse un archivo antes de terminar un programa.

Escritura en un archivo

Escribir en un archivo significa copiar un conjunto de datos almacenados en la memoria en un archivo ubicado en un dispositivo externo. La función que utilizaremos es **fwrite().** Sus parámetros son:

fwrite(&variable, tamaño_variable, cantidad_registros, punteroFILE)

El significado de cada parámetro es:

&variable: dirección de la variable que contiene la información que se quiere escribir en el archivo.

tamaño_variable: tamaño (en bytes) de la variable que contiene la información.

cantidad_registros: cantidad de registros que se quieren escribir.

punteroFILE: puntero sobre el que se abrió el archivo en el que se quiere escribir.

La función envía la información contenida entre la dirección de memoria &variable hasta &variable+ (tamaño_variable* cantidad_registros) al flujo al que apunta punteroFILE; esta información se escribe en el archivo, y el puntero FILE es desplazado una cantidad de bytes equivalente a la información escrita.

Veamos un ejemplo:

```
class Articulo
  {
private:
```

```
    char cod[5];
     char desc[30];
    float pu;
    int stock;
public:
 //métodos
    void Cargar();
    void Mostrar();

};
```

Nota: la clase Articulo se desarrolló de manera completa en otro ejemplo.

```
int main()
{
    FILE *part;
    Articulo reg;
    part=fopen("articulo.dat","wb"); //Se abre en modo escritura
    if(part==NULL)// Se comprueba que la apertura fue correcta
    { cout<<"Error de archivo";
    exit(1);    }

    reg.Cargar();////Ingreso de los datos

    fwrite(&reg, sizeof reg,1,part); //Escritura del archivo
    fclose(part);// Cierre del archivo
    return 0;
}
```

En el ejemplo se abre un archivo binario en modo escritura, se asignan valores a cada una de las propiedades del objeto Articulo, se escribe el registro en el archivo, y luego se lo cierra. Veamos la llamada a fwrite()

fwrite(&reg, sizeof reg,1,part)

&reg: dirección del objeto reg. Éste contiene la información que se quiere escribir en el disco.

sizeof reg: tamaño de la variable objeto reg. En lugar de escribir directamente el número de bytes (que para el caso del Articulo sería 5+30+4+4=43. Ver nota aparte), lo cual podría

inducirnos a cometer errores, utilizamos el operador sizeof. Este operador devuelve la cantidad de bytes de una variable o de un tipo de datos. Por ejemplo:

```
int entero;
cout<<sizeof entero; //se imprime por pantalla el tamaño de la variable
entero
cout<<sizeof(int); //se imprime por pantalla el tamaño del tipo de dato
int
```

Cuando se utiliza con una variable no es necesario usar paréntesis; cuando se utiliza con un tipo de datos deben utilizarse paréntesis entre el operador y el tipo de datos.

Nota: al declarar un objeto no siempre su tamaño es igual a la suma del tamaño de sus propiedades. Esto se debe a detalles de organización en la memoria, al utilizar propiedades de distinto tipo.

1: sólo deseamos escribir un registro.

part: puntero FILE sobre el que se hizo la apertura del archivo

En el ejemplo la función envía la información contenida entre la dirección de memoria &reg hasta &reg+ sizeof reg* 1 (es decir toda la información contenida en reg) al flujo al que apunta part; esta información se escribe en el archivo articulo.dat.

Si fuera necesario podríamos escribir de una sola vez varios registros. Por ejemplo:

```
void escribir_archivo(Articulo * v)
{
  FILE *part;
  part=fopen("articulo.dat","ab");
  if(part==NULL)
    { cout<<"Error de archivo";
      exit(1);
      }
  fwrite(v, sizeof(Articulo),5,part);
  fclose(part);
}
```

La función recibe la dirección de inicio de un vector de artículos de 5 componentes previamente cargado. La función fwrite() envía la información contenida desde la dirección &v[0] (que es lo mismo que escribir v), hasta la dirección v+sizeof(struct articulo)*5, o sea todo lo que contiene el vector. Nótese que para el segundo parámetro se utiliza sizeof(Articulo), ya que en este caso no sería correcto usar sizeof v. ¿Por qué? v es un

puntero, y como todos los punteros su tamaño es de 4 bytes. Este es uno de errores más frecuentes y difíciles de encontrar, ya que el compilador no dará ningún mensaje de error.

Como se dijo antes, C no controla ni el formato, ni la "calidad" de los datos que se envían. Esto lo podemos ver en la función fwrite() ya que lo único que ella exige es lo siguiente:
1º parámetro: una dirección de memoria
2º y 3º parámetro: un número
4º parámetro: un puntero FILE.

Si se cumple con eso, el compilador no tendrá ninguna objeción de ejecutar el código, aunque lo que se grabe sea basura.

También se mencionó más arriba que los archivos que utilizaríamos estarían compuestos por un conjunto de registros de longitud fija, por lo que la única garantía de escribir archivos de esa característica –y luego recuperar lo que escribimos sin ninguna diferencia- **es utilizar tanto para la escritura como para la lectura variables objeto de una misma clase.**

Por último, tanto el nombre como la extensión utilizada son arbitrarios. Podemos asignarle cualquier nombre y cualquier extensión, con la precaución de evitar nombres o extensiones que sean utilizadas por el sistema operativo u otras programas, para evitar confusiones.

Lectura de un archivo

Leer de un archivo significa copiar un conjunto de datos almacenados en un archivo ubicado en un dispositivo externo, y luego escribirlos a la memoria. La función que utilizaremos es **fread().** Sus parámetros son:

fread(&variable, tamaño_variable, cantidad_registros, punteroFILE)

Los parámetros son los mismos que los de fwrite(), pero su significado es distinto:

&variable: dirección de la variable donde se desea escribir la información que se leyó del archivo.

tamaño_variable: tamaño (en bytes) de la variable donde se escribirá la información.

cantidad_registros: cantidad de registros que se quieren leer del archivo.

punteroFILE: puntero sobre el que se abrió el archivo que se quiere leer.

La función lee desde la posición a la que apunta punteroFILE en el archivo una cantidad de bytes equivalente a (tamaño_variable* cantidad_registros); esta información se escribe en la memoria a partir de la dirección &variable, y el puntero FILE es desplazado una cantidad de bytes equivalente a la información leída.

Veamos un ejemplo:

```
int main()
{
    FILE *part;
    Articulo reg;
    part = fopen("articulo.dat", "rb"); // Apertura en modo lectura
     if (part == NULL)                  // Se comprueba que la apertura
fue correcta
    {
        cout << "Error de archivo";
        exit(1);
    }
    fread(&reg, sizeof reg, 1, part); // Lectura del archivo
    fclose(part);                    // Cierre del archivo
                                      // Impresión de los datos leídos
del archivo y escritos en la  //variable Articulo
    reg.Mostrar();
    // Fin de impresión de datos
    return 0;
}
```

En el ejemplo se abre un archivo binario en modo lectura, se lee un registro del archivo, luego se cierra el archivo, y por último se muestran los datos contenidos en reg. Veamos la llamada a fread()

fread(&reg, sizeof reg,1,part)

&reg: dirección de la variable reg. A partir de esta dirección se escribirá la información que se lee del disco.

sizeof reg: tamaño de la variable reg.

1: sólo deseamos leer un registro.

part: puntero FILE sobre el que se hizo la apertura del archivo

En el ejemplo la función lee sizeof reg* 1 bytes (es decir un solo registro) desde el flujo al que apunta part; esta información se escribe en la memoria a partir de la dirección &reg.

Si fuera necesario podríamos leer de una sola vez varios registros. Por ejemplo:

```
void leer_archivo(Articulo * v)
{
  FILE *part;
```

```
  part=fopen("articulo.dat","rb");
  if(part==NULL)
    { cout<<"Error de archivo";
      exit(1);
      }
  fread(v, sizeof(Articulo),5,part);
  fclose(part);
  mostrarVector(v,5);
}
```

La función recibe la dirección de inicio de un vector de artículos de 5 componentes. fread() lee del archivo los 5 primeros registros, y los escribe en la memoria a partir de la dirección &v[0] (que es lo mismo que escribir v).

Lectura secuencial de un archivo

Al leer un registro, el puntero FILE se desplaza automáticamente una cantidad de bytes equivalente al tamaño en bytes del registro (en realidad lo que se desplaza es el indicador de posición que contiene el puntero FILE), por lo que queda apuntando al registro siguiente; si no existen más registros apuntará al indicador de fin de archivo (EOF). A su vez, fread() devuelve un valor equivalente a la cantidad de registros leídos si la operación fue exitosa, por lo que es posible leer todo un archivo registro a registro, sin necesidad de conocer la cantidad de registros que contiene.

Veamos un ejemplo:

```
void listar_archivo( )
{
  FILE *part;
  Articulo reg;
  part=fopen("articulo.dat","rb");
  if(part==NULL)
    { cout<<"Error de archivo";
      exit(1);
      }
  while(fread(&reg, sizeof(Articulo),1,part)==1)
     {
        reg.Mostrar();
     }
}
```

La función lee un registro del archivo, lo almacena sobre la variable reg, y lo muestra. La lectura se hace dentro de un while en el que se compara el valor que devuelve fread() contra un 1; como se está leyendo 1 registro por vez, fread() devolverá un1 si pudo leer; en caso que no existan más registros para leer (se llegó al final del archivo), la condición del while se convertirá en falsa y el ciclo se cortará.