

# Programación de estructuras lineales

Tema 6. Los archivos

## Tema 6. Los archivos

### Índice

Objetivos de aprendizaje .....	3
1. Presentación.....	3
2. Los archivos .....	3
3. Trabajar con archivos .....	4
4. El manejo de archivos en C++ .....	5
5. Lectura de un archivo de texto.....	6
6. Escritura en un archivo de texto .....	7
7. Trabajar con registros y archivos de texto .....	8
8. Los archivos binarios .....	9
8.1. Ejemplos de archivos binarios de acceso secuencial .....	10
9. Resumen .....	12
Referencias bibliográficas .....	12

## Tema 6. Los archivos



### Objetivos de aprendizaje

Los objetivos que se pretenden alcanzar en este recurso son los siguientes:

- Qué son los **archivos** y cómo se clasifican.
- Los flujos de datos en **C++**.
- Cómo leer y escribir en **archivos de texto**.
- Cómo leer y escribir en **archivos binarios**.

### 1. Presentación

Hasta ahora, para trabajar con los datos sobre los que actuaba el programa hemos utilizado estructuras de datos como los *arrays*. Estos datos se almacenan en memoria, y cuando acaba el programa se pierden, ya que la memoria es volátil. Cuando necesitemos guardar los datos de manera más permanente, necesitaremos almacenarlos en un dispositivo de almacenamiento externo como las memorias externas.

### 2. Los archivos

Un archivo o fichero se define como un conjunto de información del mismo tipo tratado como una **unidad de almacenamiento** y organizado de forma **estructurada** para la recuperación **de un elemento o dato individual**. Los programas de ordenador normalmente trabajan con datos almacenados en la memoria principal. Trabajar con la memoria principal es muy eficiente en tiempo de acceso, pero tiene el inconveniente de que es volátil y limitada. Por ello, los ordenadores almacenan su información en **dispositivos de almacenamiento**.



Los **dispositivos de almacenamiento** son los discos magnéticos (discos duros), ópticos (CD y DVD) o memorias *flash*. Estos dispositivos disponen de una gran capacidad de almacenamiento y son gestionados directamente por el sistema operativo de la máquina que los organiza mediante archivos.

- 
- Ventajas:
    - Permite trabajar con un volumen mayor de datos.
    - Los datos no se pierden al terminar el programa.
    - Los datos pueden compartirse entre distintas aplicaciones.

## Tema 6. Los archivos

- Inconvenientes:
  - La manipulación de los datos es más compleja.
  - Los accesos a disco son bastante más lentos que los accesos a memoria.

Si optamos por clasificar los archivos dependiendo de su contenido, podemos distinguir dos tipos:

- **Archivos de texto.** Almacenan la información en modo texto; guardan los caracteres ascii de manera que pueden visualizarse con cualquier editor de texto.
- **Archivos binarios.** La información se guarda tal y como se guardan en memoria, es decir, en binario (unos y ceros).

Dependiendo de su modo de acceso, los archivos se clasifican en:

- **Archivos secuenciales**
  - Los datos se almacenan de manera consecutiva, es decir, para acceder al dato  $n$  antes hay que pasar por los  $n-1$  datos anteriores, igual que se haría con una cinta magnética. Los archivos de texto son de acceso secuencial.
- **Archivos de acceso aleatorio**
  - Se puede acceder a un dato concreto sin necesidad de pasar por todos los anteriores, como se haría en un DVD.

### 3. Trabajar con archivos

Cuando un programa quiere trabajar con archivos almacenados en memoria secundaria, normalmente hay que seguir el siguiente proceso:

- Incluir las librerías necesarias.
- Declarar una variable de tipo archivo adecuada.
- Abrir el archivo. Esta operación establece un vínculo entre la variable declarada en nuestro programa y el archivo físico gestionado por el sistema operativo.
- Comprobar que la apertura del archivo se realizó correctamente.
- Realizar la transferencia de información con el archivo a través de la variable vinculada.
- Comprobar que el procesamiento del archivo se realizó correctamente.
- Cerrar el archivo, desvinculando la variable del archivo. Si no se cierran adecuadamente, nuestro programa puede funcionar de manera imprevista. Idealmente, un destructor de la clase puede encargarse de esta operación de forma automática utilizando la técnica RAII.

## Tema 6. Los archivos

### 4. El manejo de archivos en C++

C++ ve a un archivo como un *stream*, es decir, como un flujo de *bytes*. Todos los archivos terminan con un marcador especial llamado *fin de archivo*. Hasta ahora, hemos utilizado estos flujos para leer de teclado y escribir por pantalla. C++ trata a la entrada y salida estándar como archivos de texto y crea automáticamente los objetos globales *cin* para la entrada de datos y *cout* para la salida.

**Incluir las librerías necesarias.** Para realizar el procesamiento de archivos en C++ deben incluirse las librerías necesarias. Para trabajar con archivos se utiliza la librería `<fstream>`.

```
include <iostream> //entrada salida estandar
include <fstream> //entrada salida con archivos
```

**Declarar una variable de tipo flujo de datos adecuada.** *idarchivo* es un identificador para nuestra variable archivo, no hay que confundir el nombre de la variable *stream* con el nombre del archivo en disco.

```
fstream idarchivo // declara un archivo para entrada y/o salida
ifstream idarchivo; // declara un archivo de entrada
ofstream idarchivo // declara un archivo de salida
```

**Abrir el flujo de datos** vinculando la variable declarada anteriormente con el archivo físico en disco. Esto se hace mediante el método *open*.

```
idarchivo.open(nombredelarchivofisico, modo [,modo]);
```

Donde **nombredelarchivofisico**: es una cadena de caracteres con el nombre del archivo por ejemplo “datos.txt” o “c://carpeta1/subcarpeta2//datos.txt”

Modo: se pueden combinar varios modos mediante el operador “|”.

Tabla 1. Modos.

ios::in	Abre el archivo para lectura.
ios::out	Abre el archivo para escritura. Si el archivo no existe, se crea y si existe, se sobrescribe el contenido.
ios::app	Modo <i>append</i> . Abre el archivo de texto para escribir por el final.
ios::binary	Abre un archivo binario.

**Cerrar el archivo.** Puede cerrarse el archivo de forma explícita mediante la siguiente operación:

```
archivo.close();
```

Sin embargo, es importante recordar que el destructor de la clase **fstream** se encuentra diseñado de forma que la función **close()** sea invocada automáticamente cuando el objeto abandona el ámbito donde fue declarado.

## Tema 6. Los archivos

### 5. Lectura de un archivo de texto

Vamos a analizar un ejemplo de un programa que lee un texto de un archivo de texto y simplemente lo muestra por pantalla. Para leer los caracteres utilizamos el método `get`, no podemos utilizar `>>` para la lectura del *stream*, ya que `>>` ignora el carácter blanco y los saltos de línea.

Para leer cadenas de caracteres:

- `getline(char* s, int n, char c="\n")` lee como máximo  $n-1$  caracteres del flujo de entrada, o hasta que encuentre el carácter de terminación (por defecto un final de línea) o hasta el fin de fichero.

```
/* Este es un ejemplo de lectura de un archivo de texto
 * lee el texto del archivo prueba.txt y escribe su contenido por
 * pantalla
 * Hay que asegurarse de crear el archivo con un texto cualquiera
con un
 * editor de texto */
#include <iostream>
#include <fstream>
using namespace std;

int main(){

    ifstream archivo; // declaramos el archivo de entrada
    char letra;
    archivo.open("prueba.txt"); // como archivo lo hemos
declarado como ifstream no hace falta especificar el modo

    if(!archivo){ // comprobamos que existe el archivo
        cout<< "El archivo no existe";
    }
    else {
        archivo.get(letra); /* hacemos una primera lectura para
que detecte el fin de archivo por si el archivo estuviera
vacio.*/
        while (!archivo.eof()){ //leer hasta el final del
fichero
            cout <<letra;
            archivo.get(letra); /* llamamos a la función get
que lee un carácter y lo guarda en la variable letra, si
utilizamos el operador >> habría omitido los espacios en blanco
y saltos de linea */
        }
    }
} // el archivo se cierra automáticamente en este punto
```

## Tema 6. Los archivos

### 6. Escritura en un archivo de texto



El siguiente programa escribe los diez primeros números enteros en un archivo de texto, uno por línea.

Para escribir en un archivo de texto podemos utilizar << y también el método *put* para escribir caracteres:

```
int main(){

    ofstream archivo;

    archivo.open("numeros.txt");

    if(!archivo){
        cout<< "Error creando archivo";
    }
    else{
        for(int i=0; i<10;i++){

            archivo<< i << endl; //escribimos los numeros en el
            archivo igual que lo haríamos por pantalla

        }
    } // el archivo se cierra automáticamente en este punto
```



## Tema 6. Los archivos

### 7. Trabajar con registros y archivos de texto

En este ejemplo queremos leer distintos tipos de datos de un archivo de texto. Un ejemplo de este archivo podría tener la información con el siguiente formato:

- 1 a 2.5 Luis Perez.
- 2 b -3.6 Antonio Pardo.

```
#include <iostream>
#include <fstream>
using namespace std;
const int MAX=50;

/*Este programa lee distintos tipos de datos de un archivo de
texto, uno por linea */

int main(){

    //Definimos una estructura con los campos
necesarios
        // un entero, una letra, un float y una cadena de
caracteres

    struct {

        int numero;
        char letra;
        float decimal;
        char nombre[MAX];
    }ficha;

    fstream archivo;

    archivo.open("prueba2.txt", ios::in); //si archivo es una
variables fstream es necesario especificar el modo lectura

    if(!archivo){
        cout<< "Error al leer archivo";
    }
    else{

        archivo>> ficha.numero;
        archivo>>ficha.letra;
        archivo>>ficha.decimal;
        archivo.getline(ficha.nombre, MAX); // si utilizaramos
un >> tendríamos problemas con los espacios en blanco
    }
}
```

## Tema 6. Los archivos

```

        while (!archivo.eof()){

            cout<< ficha.numero<<" " << ficha.letra <<
            "<<ficha.decimal<< ficha.nombre<<endl; //escribimos el registro
            leido por pantalla

            archivo>> ficha.numero;
            archivo>>ficha.letra;
            archivo>>ficha.decimal;
            archivo.getline(ficha.nombre, MAX); //si no ponemos
            el caracter de terminación lee hasta final de linea '\n'
        }
    }
}

```

## 8. Los archivos binarios

Dependiendo del contenido de los archivos, podemos diferenciar archivos de texto y archivos binarios.

- **Archivos de texto.** Almacenan la información en modo texto, de modo que guardan caracteres. El acceso de los archivos de texto es siempre secuencial.
  - Ventajas:
    - Se pueden editar con un editor de texto como el *block* de notas.
    - Es más fácil compartir la información entre programas.
  - Inconvenientes:
    - Su manipulación es más complicada si la cantidad y el tipo de datos es complejo.
- **Archivos binarios.** La información se guarda tal y como se guardan en memoria, es decir, con 0 y 1. Pueden ser de acceso secuencial o acceso directo.
  - Ventajas:
    - Su manejo es más sencillo ya que podemos escribir y leer grupos de *bytes* más complejos de una sola vez.
  - Inconvenientes:
    - No se puede editar desde fuera, necesitamos un programa para interpretar dichos *bytes*.

Para declarar un archivo binario hay que especificarlo en el método *open*:

```
archivo.open("nombrearchivo.txt", ios::in| ios::binary);
```

## Tema 6. Los archivos

Para leer y escribir en archivos binarios, utilizaremos los métodos *write* y *read*. Para la correcta interpretación de los datos necesitamos hacer un *casting* utilizando el operador *reinterpret\_cast*.

- *write(const char\* s, int n)* escribe *n bytes* de la cadena *s* en un flujo de salida. Para calcular el tamaño en *bytes* que tenemos que escribir utilizaremos el operador *sizeof*.
- *read(char\* s, int n)* lee *n bytes* del flujo de entrada y los deposita en la cadenas.

### 8.1. Ejemplos de archivos binarios de acceso secuencial

Vamos a ver un ejemplo del empleo de archivos binarios con acceso secuencial. El archivo va a almacenar estructuras del mismo tipo que el visto en el ejemplo de archivos de texto, crearemos el archivo, escribiremos en él varios registros y después los leeremos.

```
#include <iostream>
#include <fstream>
using namespace std;
/* ejemplo de utilización de archivos binarios con acceso
secuencial */

// El archivo va a contener estructuras de tipo registro
typedef

struct {

    int numero;
    char letra;
    float decimal;
}tipoficha;

int main(){

    tipoficha ficha, fichal, ficha2;
    fstream archivoentrada, archivosalida;

    //abrimos el archivo para escritura

    archivosalida.open("estructuras.dat", ios::out|
ios::binary); //especificamos que es un archivo binario

    if(!archivosalida)
        cout<< "Error creando archivo";
    else
    {
```

**Tema 6. Los archivos**

```

// creamos unas fichas de ejemplo
ficha1.numero=3;
ficha1.letra='s';
ficha1.decimal=3.5;

ficha2.numero=13;
ficha2.letra='a';
ficha2.decimal=-43.5;

/* el tipo de ficha es tipoficha, sin embargo, el
argumento de write es const char * para que la llamada
a write compile, necesitamos hacer un cast mediante
reinterpret_cast*/

        archivosalida.write(reinterpret_cast<const char
*>(&ficha1), sizeof(tipoficha));
        archivosalida.write(reinterpret_cast<const char
*>(&ficha2), sizeof(tipoficha));

        archivosalida.close(); // cerramos el archivo
    }

// Ahora abrimos el archivo como entrada, leeremos los datos
y los mostraremos por pantalla
archivoentrada.open("estructuras.dat", ios::in|
ios::binary);

if(!archivoentrada)
    cout<< "Error abriendo archivo";
else{
    archivoentrada.read(reinterpret_cast<char *>(&ficha),
sizeof(tipoficha));

    while (!archivoentrada.eof()){

        cout<< ficha.numero<<"-"<<ficha.letra<< " - "<<
ficha.decimal<<endl; //escribimos el contenido del registro
leido
        archivoentrada.read(reinterpret_cast<char
*>(&ficha), sizeof(tipoficha));

    }

    archivoentrada.close();
}

```

### 9. Resumen

Un archivo es una secuencia de bits almacenados en algún lugar de un dispositivo de almacenamiento externo. Se utiliza para guardar la información con la que nuestro programa puede trabajar de manera permanente. Para trabajar con archivos externos se crean objetos *fstream* y a continuación se abren con el método *open*. Después de comprobar que el archivo se ha abierto correctamente, realizamos el proceso de lectura, comprobando con cada lectura que no hemos alcanzado el final del archivo (*eof*) o escritura. Una vez procesado el archivo, lo cerraremos de forma explícita con el método *close* o permitiremos que el destructor de la clase se encargue de forma automática de dicha operación cuando el flujo salga fuera del ámbito en que fue definido.

- Qué es un archivo y cómo se clasifican.
- Cómo leer y escribir en un archivo de texto.
- Cómo leer y escribir en archivos binarios de acceso secuencial.

### Referencias bibliográficas

Ceballos Sierra, F. J. (2009). *Enciclopedia del Lenguaje C++*. Madrid: RA-MA Editorial.



© Todos los derechos de propiedad intelectual de esta obra pertenecen en exclusiva a la Universidad Europea de Madrid, S.L.U. Queda terminantemente prohibida la reproducción, puesta a disposición del público y en general cualquier otra forma de explotación de toda o parte de la misma.

La utilización no autorizada de esta obra, así como los perjuicios ocasionados en los derechos de propiedad intelectual e industrial de la Universidad Europea de Madrid, S.L.U., darán lugar al ejercicio de las acciones que legalmente le correspondan y, en su caso, a las responsabilidades que de dicho ejercicio se deriven.

