

### Contenidos

1. Introducción.
2. Manipulación de ficheros.
3. Organización de archivos.
4. Tipo de almacenamiento.
5. Biblioteca de flujos.
6. Operaciones asociadas a archivos:
  - Abrir fichero,
  - Cerrar fichero,
  - Lectura y escritura,
  - Funciones de control.

### Introducción

Ya se pueden manejar gran cantidad de datos del mismo y diferente tipo al mismo tiempo (arrays y arrays de estructuras).

El problema es que el programa retiene los datos mientras esté ejecutándose y se pierden al terminar la ejecución.

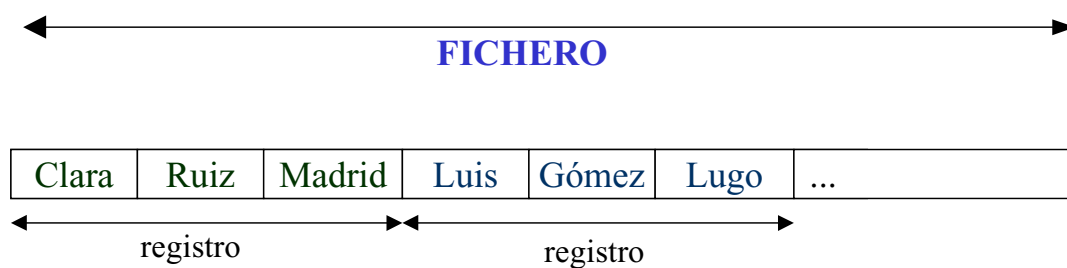
La solución para hacer que los datos no se pierdan es almacenarlos en un fichero o **archivo**.

Los **archivos** son medios que facilita el lenguaje para almacenar los datos en forma permanente, normalmente en los dispositivos de almacenamiento estándar.

## Introducción

Desde el punto de vista informático, un fichero es una colección de información que almacenamos en un soporte magnético para poder manipularla en cualquier momento.

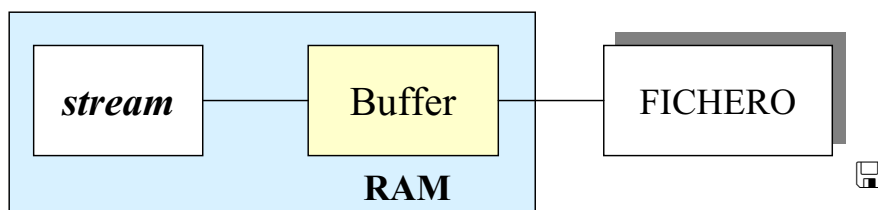
Esta información se almacena como un conjunto de **registros**.



## Manipulación de Ficheros

En C++, se utilizan streams (flujos) para gestionar la lectura y escritura de datos. Ya conocemos dos flujos estándar: **cin** y **cout**.

En definitiva, **abrir un fichero** significa **definir un stream**. Dicho stream permite la transferencia de datos entre el programa y el fichero en disco.



El buffer es un área de memoria situada en la RAM asignada al programa que abre el archivo.

### Manipulación de Ficheros

Toda transferencia de datos entre el programa y el fichero en disco se realiza a través del buffer. El buffer está para dar eficiencia.

Las operaciones de E/S son más eficientes:

- El acceso a la memoria RAM consume menos tiempo que el acceso a un dispositivo físico.
- El buffer hace que el número de accesos al fichero físico sea menor.

El uso del buffer permite realizar operaciones de entrada salida de forma más eficiente.

### Organización de archivos

Nos centraremos solo en archivos de acceso secuencial

**Archivos de acceso secuencial:** los datos se almacenan de forma consecutiva y no es posible leer un registro directamente, es decir para leer el registro  $n$  hay que leer los  $n-1$  registros anteriores.

**Archivos de acceso aleatorio:** se puede acceder a un registro concreto sin necesidad de leer todos los anteriores.

### Tipo de almacenamiento en Archivos

Nos centraremos solo en archivos de texto

**Archivos de texto:** Los datos se almacenan usando código ASCII y por tanto, pueden ser procesados por cualquier editor de texto.

**Archivos binarios:** Los datos se almacenan en binario.

## Biblioteca de flujos

Para poder manipular archivos, C++ dispone de la biblioteca estandar *fstream* (file stream) donde se encuentran todas las funciones necesarias para abrir y cerrar archivos, así como para realizar las operaciones de lectura y escritura de datos en archivos.

```
# include <fstream.h>
```

## Operaciones asociadas a archivos

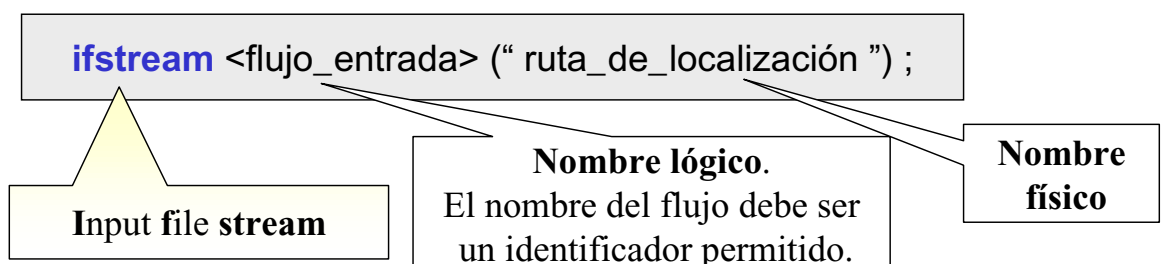
- Abrir fichero {
  - Para entrada o lectura
  - Para salida o escritura {
    - en modo truncado
    - en modo añadir
- Cerrar fichero
- Lectura y escritura
- Funciones de control

## Apertura de archivos

Al igual que los flujos **cin** y **cout**, los flujos de E/S solo pueden transferir datos en una dirección, esto significa que se tienen que definir flujos diferentes para lectura y escritura de datos.

### ➡ Abrir el archivo para lectura o entrada

Abrir un archivo para entrada, es definir un flujo de archivo de entrada.



En el programa, nos referiremos al fichero utilizando el nombre del flujo.

## Abrir el fichero para lectura: Ejemplo

Supongamos que queremos abrir un fichero que se llama *misdatos.txt* que se encuentra en la unidad de disco *a*:

```
#include <fstream.h>

....
ifstream leer_fich ("a:\\misdatos.txt");
...
...
```

Hemos definido el flujo de entrada **leer\_fich**.

## ➡ Abrir el archivo para escritura o salida

Abrir un archivo para salida, es definir un flujo de archivo de salida.

Existen dos posibilidades:

Output file stream

Nombre físico

```
ofstream <flujo_salida> (" ruta_de_localización " );
```

```
ofstream <flujo_salida> (" ruta_de_localización ", ios_base::out );
```

Nombre lógico.

El nombre del flujo debe ser un identificador permitido.

- ▶ Si se abre un archivo en modo salida y dicho archivo ya existe, todos los datos almacenados serán sobrescritos.
- ▶ Si el archivo no existe, se creará.

### Abrir el fichero para escritura: Ejemplo

Supongamos que queremos abrir un fichero para escribir datos. Dicho fichero se llama *misdatos.txt* y se encuentra en la unidad de disco *a*:

```
#include <fstream.h>
....
ofstream fich_1 ("a:\\misdatos.txt");
ofstream fich_dos ("a:\\misdatos.txt", ios_base::out);
...
```

Hemos definido el flujo de salida **fich\_1**.

Hemos definido el flujo de salida **fich\_dos**.

### ➡ Abrir el archivo para añadir datos al final

Si se desea añadir en lugar de reemplazar los datos de un archivo existente, se debe definir un flujo de salida en modo *append*.

```
ofstream <flujo_salida> (" ruta_de_localización ", ios_base::app );
```

- ▶ Los datos adicionales escritos en el archivo, se añaden en su extremo final.
- ▶ Si el archivo no existe, se creará.

```
#include <fstream.h>
....
ofstream fich_tres ("a:\\misdatos.txt", ios_base::app);
...
```

Hemos definido el flujo de salida **fich\_tres** en modo añadir.

### Advertencia

Antes de seguir adelante, es aconsejable comprobar si el fichero se ha abierto correctamente, tanto si es para lectura, escritura.

```
if (fichero_uno)
    cout << "Apertura con éxito";
else
    cout << "No se ha podido abrir el fichero";
```

Una vez definidos los flujos con los que va a trabajar nuestro programa, se pueden utilizar los operadores de inserción(<<) y extracción(>>) y los métodos ya conocidos de lectura de datos vistos para los flujos **cin** y **cout**.

### Cierre de archivos

Cuando el programa ha terminado de manipular el fichero, éste debe cerrarse. Para cerrar un archivo, basta con ejecutar la función **close** sobre el flujo asociado al fichero.

Nombre lógico

```
<nombre_flujo>. close();
```

```
#include <fstream.h>
....
ifstream leer_fich ("a:\\misdatos.txt");
ofstream fich_1 ("a:\\misdatos.txt", ios_base::out);
ofstream fich_dos ("a:\\misdatos.txt", ios_base::app);
....
leer_fich.close();
fich_1.close();
fich_dos.close();
....
```

Si un fichero no se cierra, es cerrado automáticamente cuando termina el programa


## Lectura de archivos de texto

La lectura de un archivo de texto se puede realizar con el operador de extracción (>>).

```
#include <fstream.h>
....
ifstream leer_fich ("a:\\misdatos.txt");
if ( ! leer_fich)
    cout << "No se ha podido abrir el fichero";
else
{
    leer_fich >> numero;
    leer_fich >> nombre;
}
....
leer_fich.close();
...
```

Diagram illustrating the code flow with annotations:

- Apertura del fichero (points to `ifstream leer_fich ("a:\\misdatos.txt");`)
- Comprobar que se ha abierto correctamente (points to `if ( ! leer_fich)`)
- Manipular el fichero (points to `leer_fich >> numero;` and `leer_fich >> nombre;`)
- Cierre (points to `leer_fich.close();`)

Output example:  123 Ana

El operador >> lee hasta un blanco y omite los blancos.

## Lectura de archivos de texto

La biblioteca estándar **fstream.h** nos proporciona funciones para el control de flujos. Ya las habíamos visto cuando analizamos el flujo estándar cin.

1. Lectura de cadenas, incluyendo caracteres en blanco:

**<flujo\_entrada>.getline:** *cadena* × *tamaño* × *carácter* → *void*

2. Lee un carácter del flujo de entrada y devuelve falso cuando se ha alcanzado fin de fichero

**<flujo\_entrada>.get :** *char* → *bool*

## Escritura de archivos de texto

La escritura de un archivo de texto se puede realizar con el operador de inserción (<<).


```

#include <fstream.h>
....
ofstream write_fich ("a:\\misdatos.txt");
if ( ! write_fich)
    cout << "No se ha podido abrir el fichero";
else
{
    write_fich << numero;
    write_fich << " ";
    write_fich << nombre;
}
....
leer_fich.close();
...

```

Diagram annotations:

- Apertura del fichero (points to `ofstream write_fich`)
- Comprobar que se ha abierto correctamente (points to `if ( ! write_fich)`)
- Manipular el fichero (points to `write_fich << numero;`)
- Escribe un espacio en blanco (points to `write_fich << " ";`)
- Cierre (points to `leer_fich.close();`)

Output: 123 Ana 

## Escritura de archivos de texto

Escribe un carácter en el flujo de salida.

**<flujo\_salida>.put : *char* → *void***

```

#include <fstream.h>
....
ifstream leer_fich ("a:\\datos1.txt");
ofstream write_fich ("a:\\datos2.txt");
...
leer_fich.get(car);
leer_fich.getline(nombre, 20);
write_fich.put(car);
....
leer_fich.close();
write_fich.close();
...

```

Diagram annotations:

- Lee un carácter del flujo de entrada (points to `leer_fich.get(car);`)
- Escribe un carácter del flujo de salida (points to `write_fich.put(car);`)

## Funciones de control de flujos de archivos

La función **eof**, devuelve un valor distinto de cero si el flujo de entrada ha alcanzado el final del fichero:

**<flujo\_entrada>.eof:** → *int*

```
#include <fstream.h>
....
ifstream leer_fich ("a:\\datos1.txt");
...
while (! leer_fich.eof() )
{
    leer_fich.getline(nombre, 20);
    linea ++;
}
....
leer_fich.close();
...
```

Mientras no sea final de fichero