



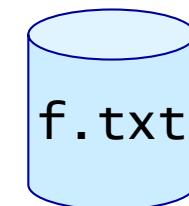
UNIVERSIDAD  
DE MÁLAGA

# PROGRAMACIÓN II

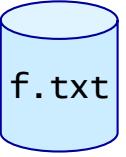
## Tema I

## Ficheros

- 1. Almacenamiento externo**
- 2. Ficheros de texto y binarios**
- 3. Programación con ficheros**
- 4. Ficheros de texto en C++**



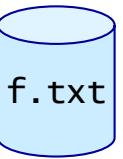
*Tema 1. Ficheros  
febrero de 2017*



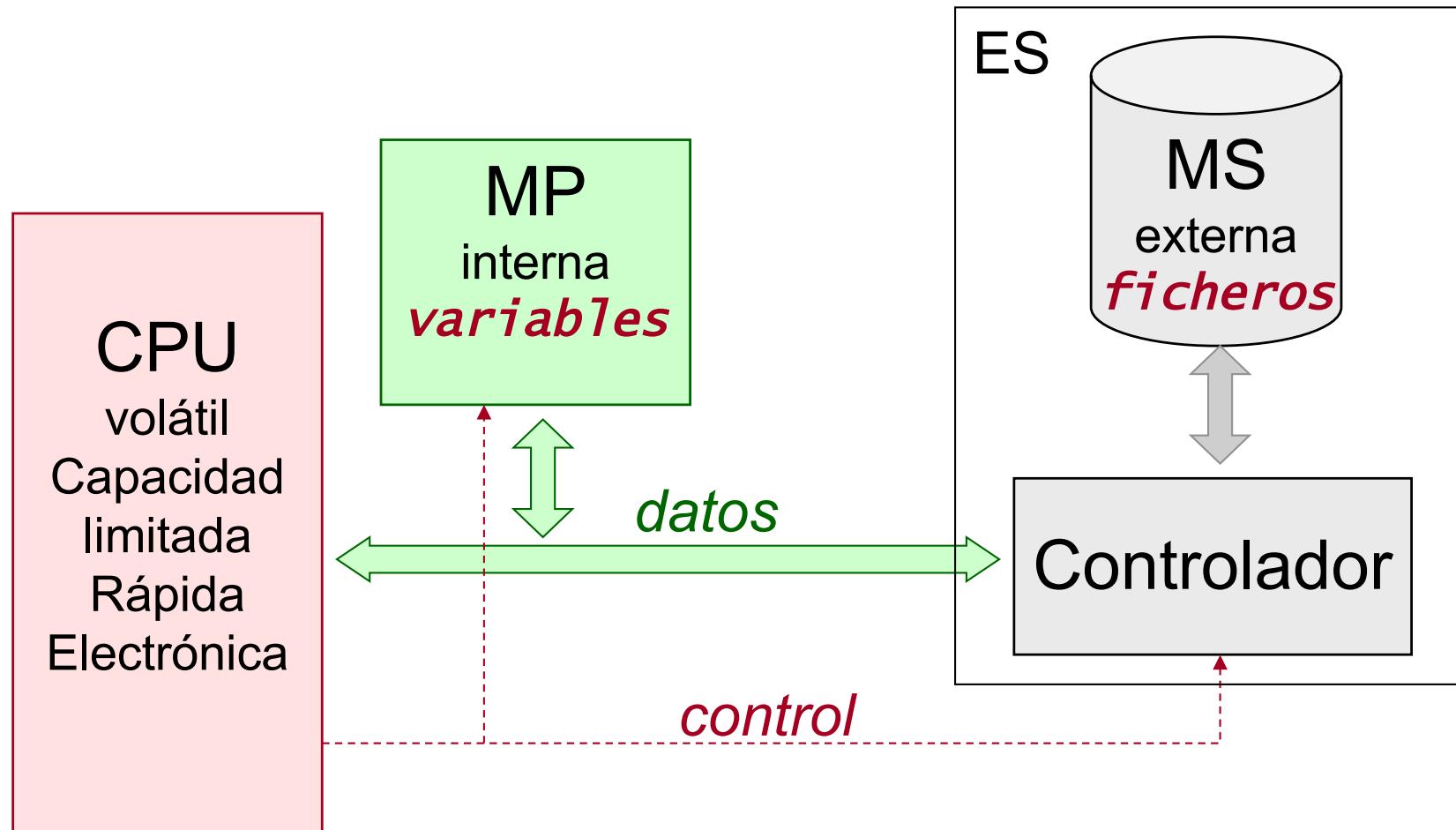
# Almacenamiento externo

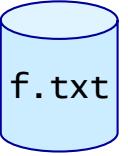
- **Necesidad**
  - ¤ Arranque ordenador y carga del S.O.
  - ¤ Datos permanentes
- **Solución: memoria secundaria**
  - + No volátil
  - + Coste bajo
  - + Alta capacidad
  - Tecnología no electrónica: magnética, óptica
  - Velocidad baja

# Almacenamiento externo



- Hardware involucrado





# Almacenamiento externo

- **Buffer**

*Área de MP del SO donde temporalmente se almacena la información transferida de MS*

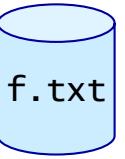
- **Bloque o registro físico**

*Unidad de transferencia de información entre la MS y la CPU (o MP)*

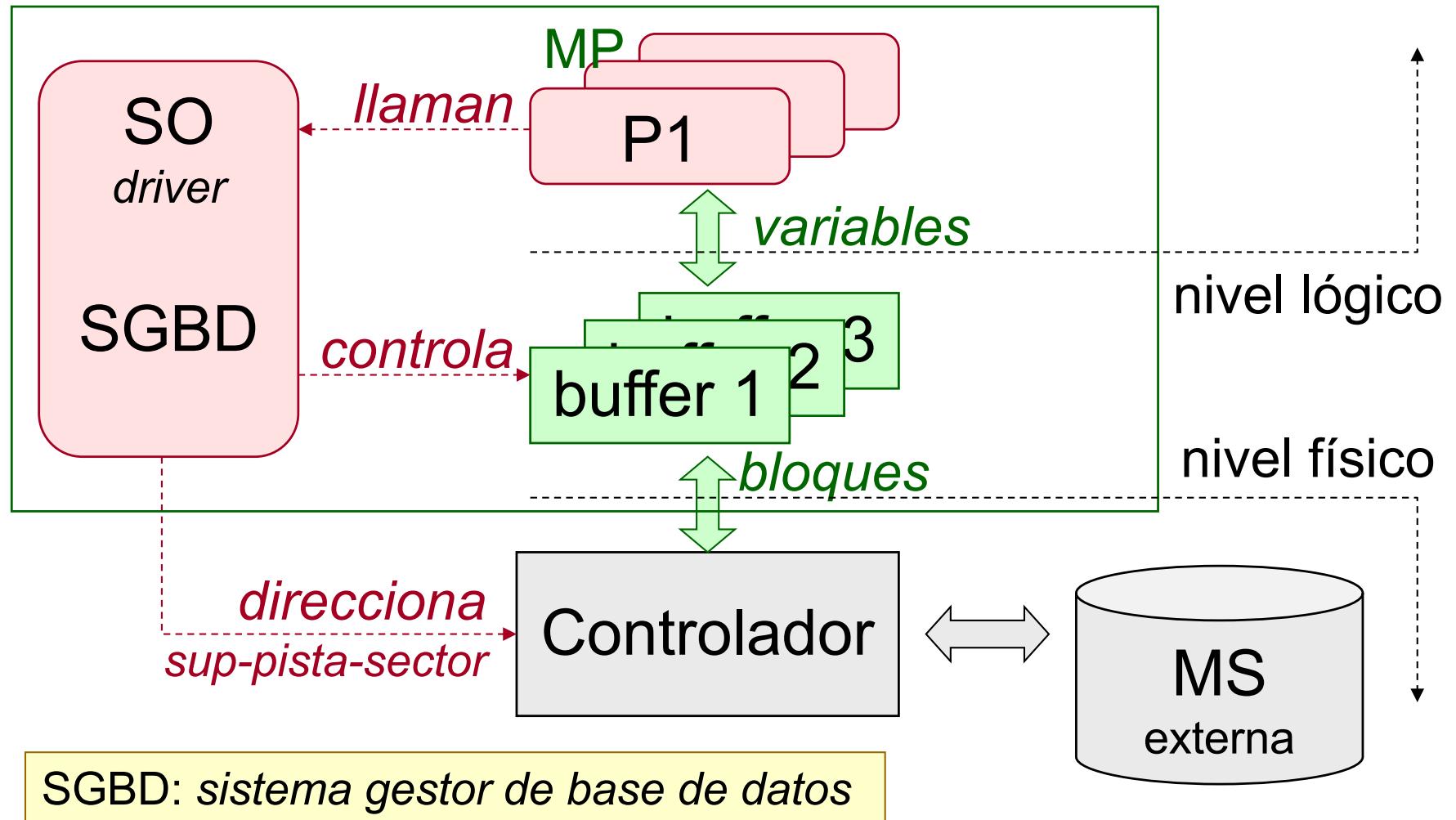
- **Driver**

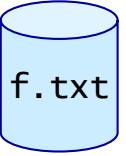
*Programa del SO que dialoga con el controlador de MS*

# Almacenamiento externo



- Papel del sistema operativo





# Almacenamiento externo

- **Fichero**

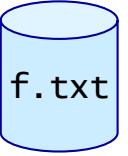
*Unidad fundamental de almacenamiento en MS, que facilita su acceso rápido y directo*

- **Operaciones sobre ficheros**

creación, destrucción, copia, clasificación  
reorganización, fusión, partición

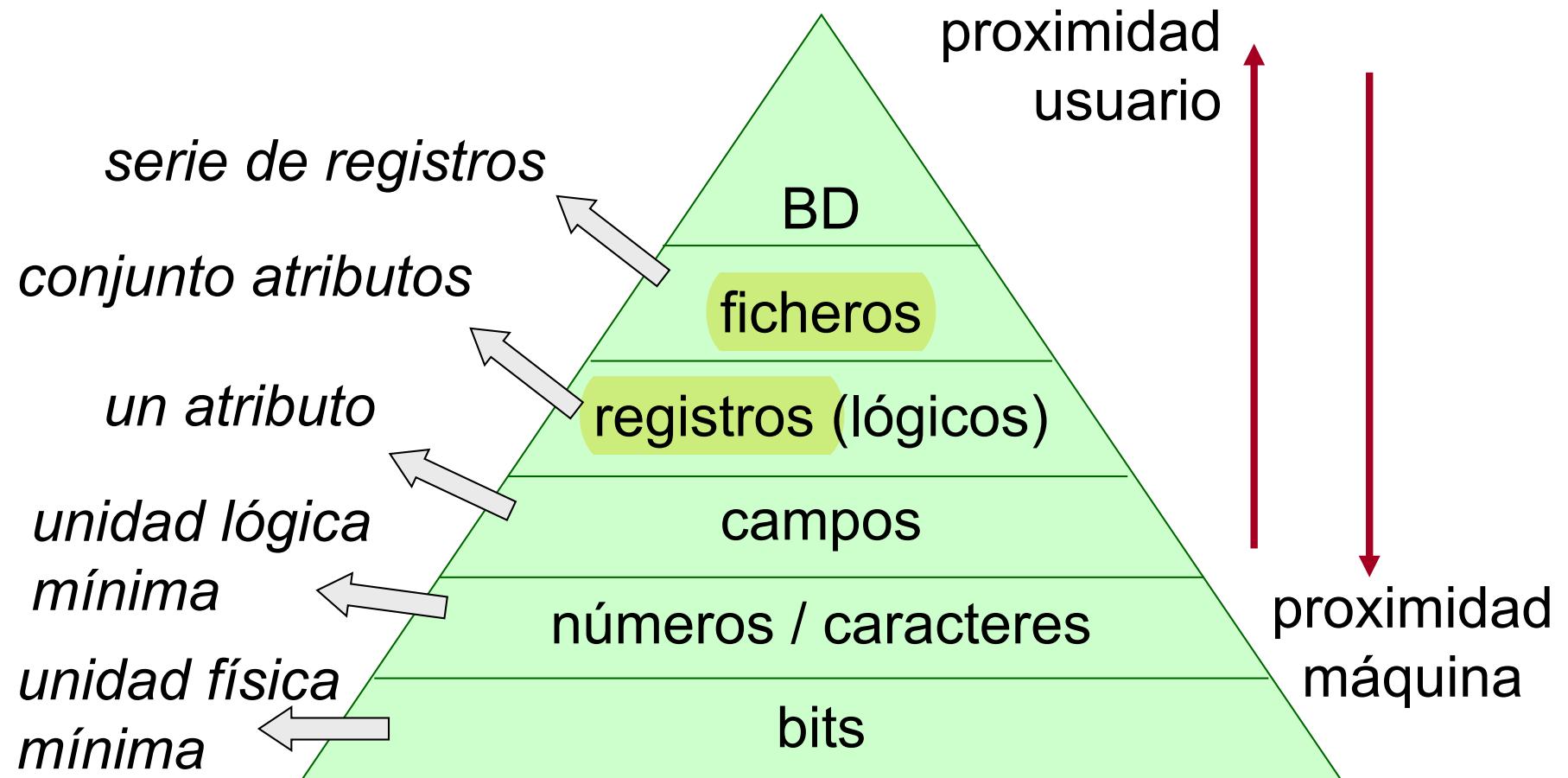
- **Operaciones sobre registros**

consulta, alta, baja, modificación



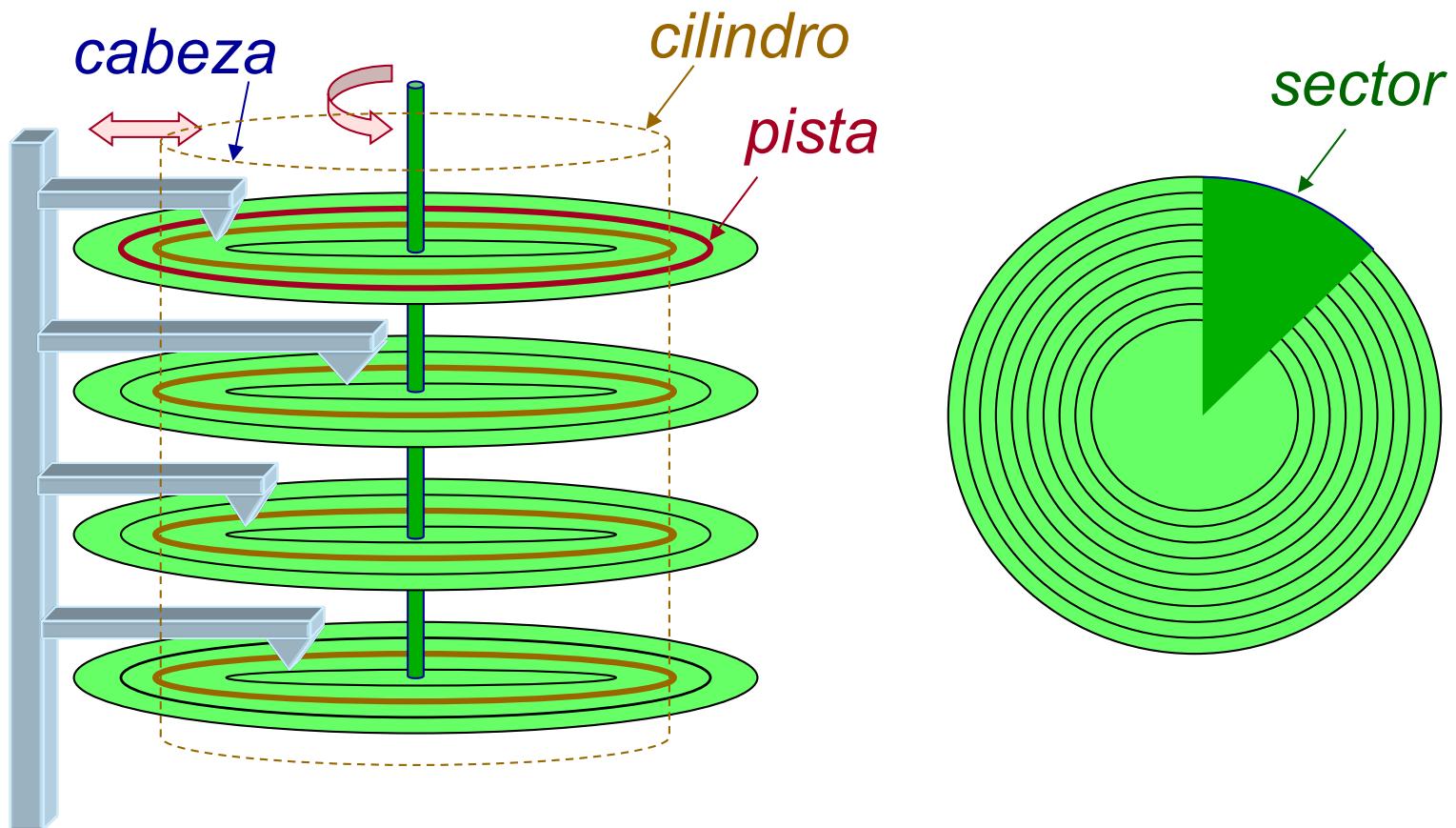
# Almacenamiento externo

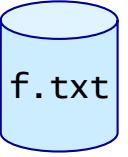
- Jerarquía de datos



# Almacenamiento externo

- Estructura física





# Almacenamiento externo

- Parámetros físicos

- ¤ *Tiempo de acceso:* segundos

$$t_{acc} = t_{loc} + t_{cab} + t_{lat} \rightarrow \text{sector (rotación)}$$

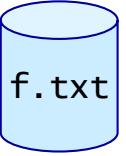
$$t_{cab} \ll t_{lat} < t_{loc}$$

cabeza (selección)

pista (despl. brazo)

- ¤ Normalmente:

- ¤ *Velocidad de transferencia:* bits/seg (bps)
  - ¤ *Capacidad:* mbytes, gbytes, tbytes, pbyte, ...
  - ¤ *Densidad de grabación:* bits/pulgada (bpi)



# Almacenamiento externo

- **Tipos de ficheros según contenido**

fuente (**.cpp**), objeto (**.o**), ejecutable (**.exe**)

texto (**.txt**), datos (**.dat**), comprimido (**.zip**)

script (**.ini**, **.sh**)

documento (**.tex**, **.doc**), postscript (**.ps**)

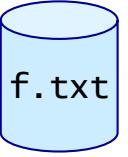
imagen (**.png**, **.jpg**, **.bmp**), sonido (**.wav**, **.mp3**)

vídeo (**.avi**, **.mpg**), hipertexto (**.html**, **.asp**)

bases de datos (**.db**, **.dbf**, **.mdb**)

hoja de cálculo (**.xlsx**, **.wg1**)

...



# Ficheros de texto y binarios

- **Tipos de ficheros según formato**

## Texto

*Codificados con conjunto de caracteres estándar (ASCII, Latino-1, Unicode)*

Ejemplos: fuente, postscript, html

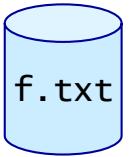
## Binario

*Formato nativo de la MP de la máquina  
(ASCII, Latino-1, Unicode)*

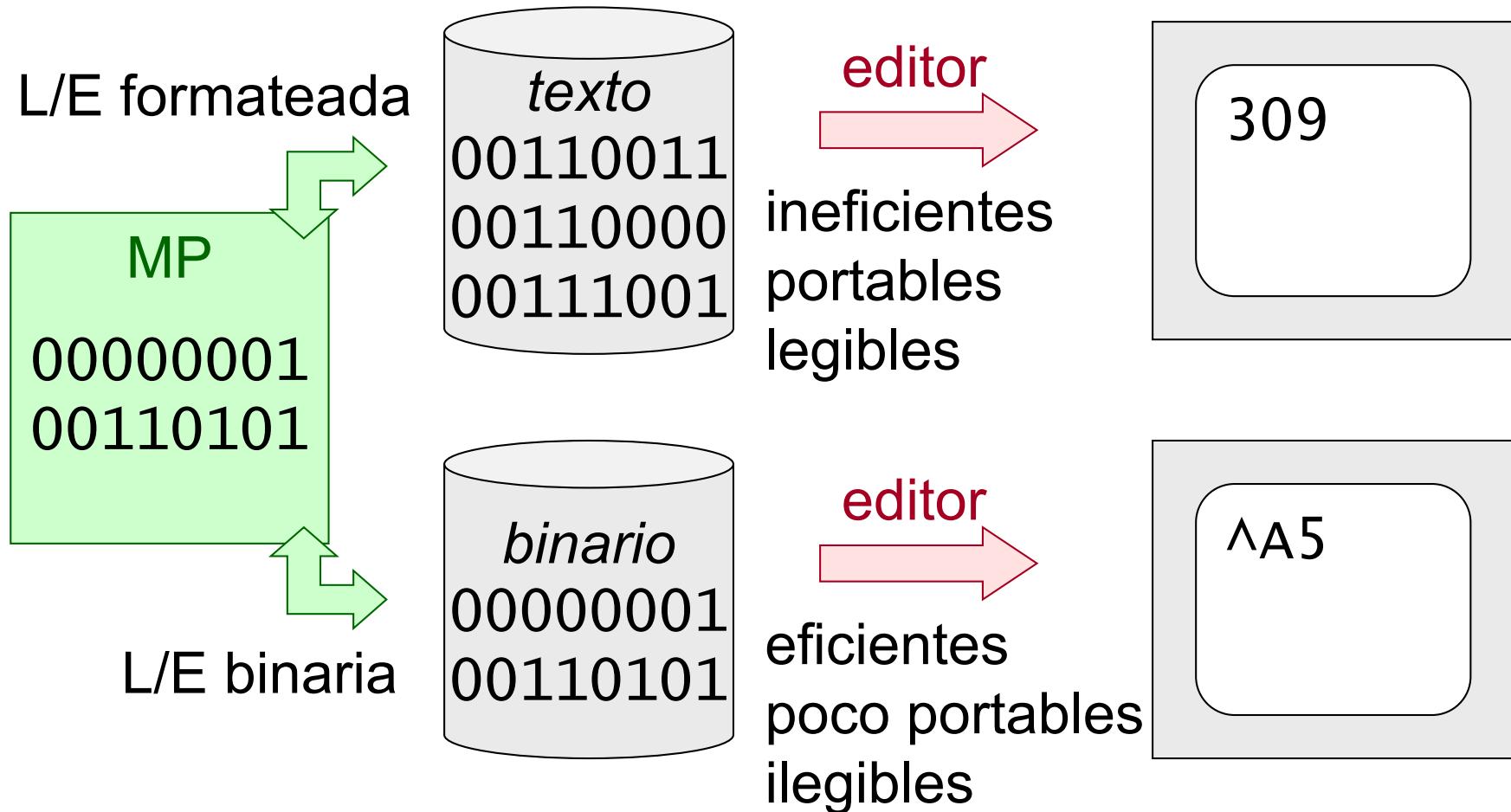
Ejemplos: ejecutable, objeto, datos

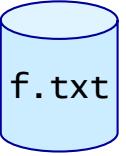
Hay formatos  
estándares  
(IEEE 754)

# Ficheros de texto y binarios



- **Fichero texto Vs. fichero binario**





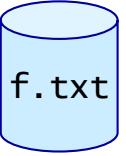
# Programación con ficheros

- **Manejador de fichero**

*Variable de tipo especial, dependiente del lenguaje de programación, que permite manejar ficheros por programa*

Contiene información para:

- ¤ acceder a los datos en MS (**nombre.ext**)
- ¤ acceder a los datos en los buffers del SO
- ¤ posición actual de lectura/escritura
- ¤ permisos, fechas, tamaño...



# Programación con ficheros

- **Apertura de un fichero**

- - ¤ Obtención del manejador e inicialización de buffers del SO

- **Cierre de un fichero**

- - ¤ Devolución de los recursos utilizados al SO

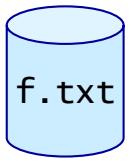
- **Lectura/escritura secuencial**

- - ¤ Actualización posición de l/e automática

- **Lectura/escritura directa**

- - ¤ Actualización posición de l/e manual

# Programación con ficheros

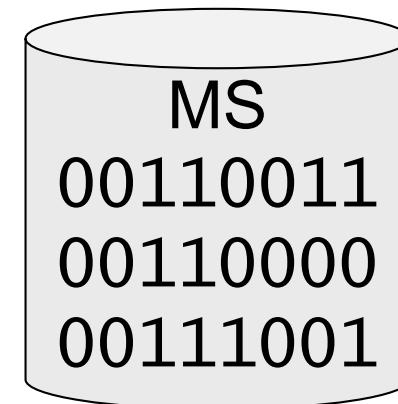
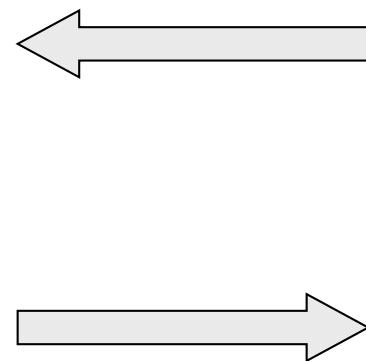
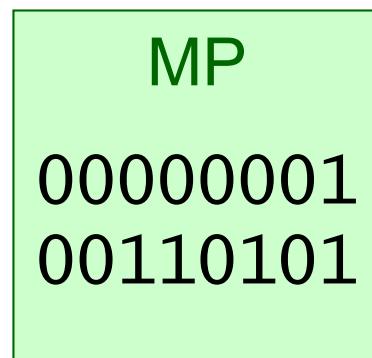


- **Lectura/escritura formateada**

Conversión formato MP nativo  $\leftrightarrow$  caracteres

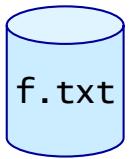
El tipo de la variable determina la conversión

Leer(E TManejador f; s *Tipo var*)



Escribir(ES TManejador f; E *Tipo var*)

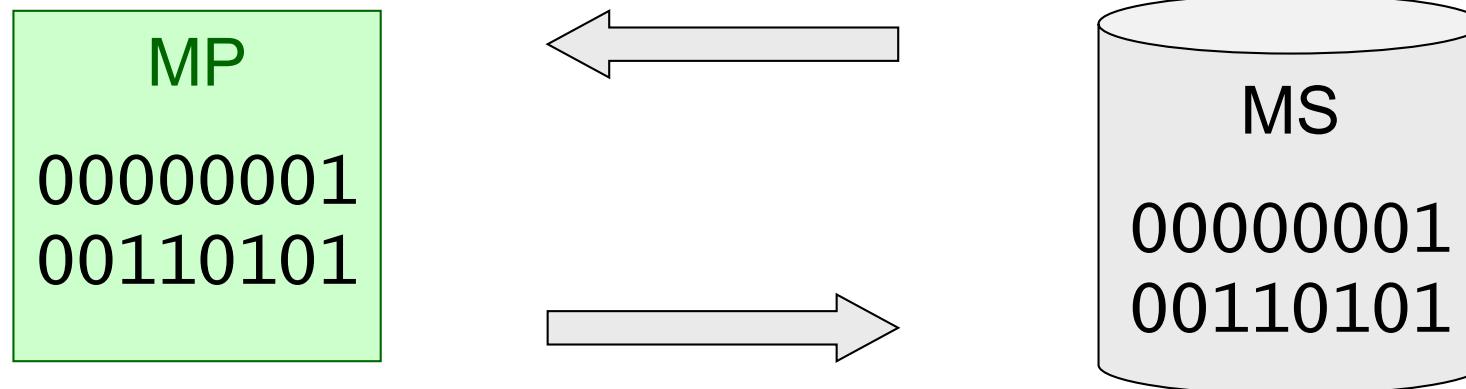
# Programación con ficheros



- **Lectura/escritura no formateada**

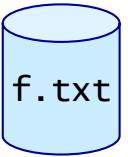
No hay conversión: sólo volcado nativo de MP  
Se leen/escriben bytes

`LeerBin( ... ; E N nbytes)`

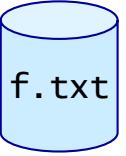


`EscribirBin( ... ; E N nbytes)`

# Flujos de Entrada/Salida en C++



- **C++ considera los ficheros como flujos de entrada/salida.**
- **Un flujo puede considerarse como una secuencia de caracteres.**
- **Flujos de entrada/salida estándar:**
  - `cin`: flujo de entrada de caracteres por teclado.
  - `cout`: flujo de salida a pantalla o monitor.
  - `cerr`: flujo de error (suele ser la pantalla).



# Declaración de flujos

- La biblioteca **fstream** define los tipos y funciones para los flujos:

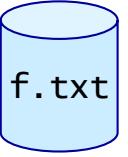
```
#include <fstream>
```

- Los flujos **cin** y **cout** ya están definidos en la biblioteca **iostream**.
- **Flujos de entrada:** de tipo **ifstream**

```
ifstream flujo_ent;
```

- **Flujos de salida:** de tipo **ofstream**

```
ofstream flujo_sal;
```

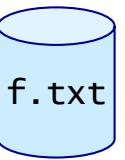


# Apertura de ficheros de texto

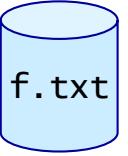
- **Un flujo tiene que estar asociado a un fichero**
  - ¤ El fichero se identifica mediante un nombre, que consiste en una cadena de caracteres.
- **Se emplea el *método open ()* :**

```
flujo_ent.open ("entrada.dat") ;  
flujo_sal.open ("salida.sal") ;
```
- **¡Cuidado! Si el flujo se abre para escritura y el fichero ya existe,**
  - ¤ se pierde su contenido, y
  - ¤ se trabaja como si el fichero estuviera vacío.

# Cierre de ficheros



- Termina la asociación entre el flujo y el fichero para dejar el flujo libre.
- Se emplea el *método close ()*:  
`flujo_ent.close();`  
`flujo_sal.close();`
- Un flujo puede volver a asociarse a un fichero tras haber cerrado la conexión con el anterior.



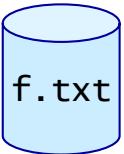
# Entrada/salida básica

- Las operaciones << y >> están disponibles para flujos de ficheros:

```
int i;  
flujo_ent >> i; // Lee de fichero  
flujo_sal << i; // Escribe en fic.
```

- Las operaciones << y >> están definidos para los tipos predefinidos de C++ y para cadenas de caracteres.

# Ejemplo de E/S con ficheros

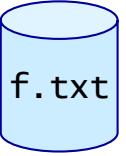


Ej. 01

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main() { // Lee 5 números enteros desde fichero
    const unsigned cant_nums = 5;
    const string fichero = "datos.in"; // Nombre del fic.
    int num;
    ifstream f_ent;
    f_ent.open(fichero.c_str()); // Añadido por el profesor
    for (unsigned i = 0; i < cant_nums; i++) {
        f_ent >> num;
        cout << num << endl;
    }
    f_ent.close();
}
```

El método `c_str()` devuelve  
una cadena de C equivalente  
a la cadena `string` de C++



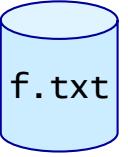
# Otras funciones sobre flujos

- **fail()**

- ❑ Se puede usar después de open() para saber si el fichero se ha abierto correctamente.
- ❑ Posibles fallos detectados por la función:
  - Apertura para lectura de un fichero que no existe.
  - Apertura para escritura de un fichero para el que no tenemos permiso de escritura.
- ❑ Informa de fallos de lectura. Ej., se lee una letra cuando se pretende leer un número.

- **eof()**

- ❑ Función booleana: indica si se ha llegado al final del fichero.

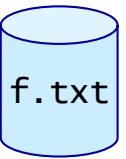


# Ejemplo: uso de fail() y eof()

Ej. 02

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main(){ // Leemos enteros hasta final de fichero
    const string fichero = "datos.in";
    int num;
    ifstream f_ent;
    f_ent.open(fichero.c_str());
    if (f_ent.fail()){
        cout << "Imposible abrir el fichero de entrada " << endl;
    }else{
        f_ent >> num; // LECTURA POR ADELANTADO PARA DETECTAR
                        // FIN DE FICHERO <-- ¡MUY IMPORTANTE!
        while (!f_ent.eof()){
            cout << num << endl;
            f_ent >> num;
        }
        f_ent.close();
    }
}
```



# Detectando final de fichero...

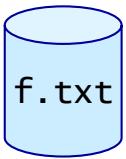
- ... por el resultado de la operación >>

```
/* cuenta números enteros de fichero de texto */
# include <iostream> // cout <<
# include <fstream> // open, close, f >>

using namespace std;
int main ()
{
    ifstream f; // manejador fichero de entrada
    int numero, nlineas=0;
    f.open("texto.txt");
    while( f >> numero ) { // false si es fin de fic.
        nlineas++;
    }
    cout << "Número líneas: " << nlineas << endl;
    f.close();
}
```

Ej. 03

# Lectura de caracteres con get()



Ej. 04

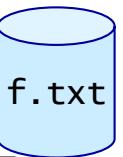
```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main() {
    const string fichero = "datoscar.in";
    char car;
    ifstream f_ent;

    f_ent.open(fichero.c_str());
    if (!f_ent.fail()) {
        f_ent.get(car);
        while (!f_ent.eof()) {
            cout << car;
            f_ent.get(car); // Si leemos con f_ent >> car se salta separadores
        }
        f_ent.close();
    }
}
```

La función equivalente para escritura es put(), que tiene un char como parámetro.

# Fichero como parámetro



Ej. 07

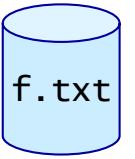
```
#include <iostream>
#include <fstream>
using namespace std;
void volcarFichero(ifstream &ff);

int main(){
    const string nombre_fichero = "cadenas.in";
    ifstream f_ent;
    f_ent.open(nombre_fichero.c_str());
    if (f_ent.fail()){
        cout << "Imposible abrir el fichero de entrada" << endl;
    }else{
        volcarFichero(f_ent);
        f_ent.close();
    }
}

void volcarFichero(ifstream &ff){
    char cc;
    ff.get(cc);
    while (!ff.eof()){
        cout << cc;
        ff.get(cc);
    }
}
```

Los flujos se pasan  
siempre como parámetro  
por referencia (&)

# Lectura de cadenas con getline



```
#include <iostream>
#include <fstream>
using namespace std;

void leer_texto(ifstream &f, string &texto);

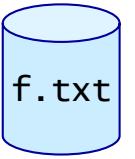
int main(){
    string texto;
    ifstream f;
    f.open("poema.txt");
    if (!f.fail()){
        leer_texto(f, texto);
        f.close();
    }
    cout << endl << texto << endl;
}

void leer_texto(ifstream &f, string &texto){
    string linea;
    getline(f, linea); // Linea 1
    while (!f.eof()) {
        texto += linea + '\n';
        getline(f, linea); // Linea 2
    }
}
```

ejemplo\_getline.cpp

Le pasamos como parámetros el flujo de entrada y el string donde queremos leer.

# Ejemplo de escritura a fichero



```
#include <iostream>
#include <fstream>

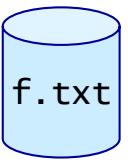
using namespace std;

int main () {
    ofstream f; // Flujo de salida
    const string nom_fic = "datos_salida.txt";
    f.open(nom_fic.c_str());
    if (!f.fail()) {
        f << "Ana Sanchez Gomez" << endl;
        f << "25555555W" << endl;
        f << 4 << ' ' << 12 << ' ' << 2004 << endl;
        f.close();
    } else{
        cout << "No puede abrirse el fichero" << endl;
    }
}
```

ejemplo\_escritura.cpp

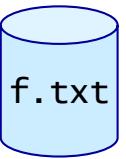
Es necesario separar los datos mediante separadores: espacios, tabuladores, saltos de línea.

# Escribir un array en fichero



```
#include <iostream>
#include <fstream>
#include <array>
using namespace std;
typedef array<int, 10> Datos;
void escribir_fichero(ofstream &f, const Datos &datos);
int main () {
    ofstream f;
    Datos vector = {{0,1,2,3,4,5,6,7,8,9}};
    const string nom_fic = "datos_salida.txt";
    f.open(nom_fic.c_str());
    if (!f.fail()){
        escribir_fichero(f, vector);
        f.close();
    } else{ cout << "No puede abrirse el fichero" << endl;}
}
void escribir_fichero(ofstream &f, const Datos &datos) {
    for (int i = 0; i < int(datos.size()); i++){
        f << datos[i] << endl; //Separador: salto de línea
    }
}
```

escribir\_array.cpp



# Bibliografía

- [JOYA03] Joyanes, L. (2003), Fundamentos de programación: algoritmos, estructuras de datos y objetos, McGraw-Hill.
- [JOYA00] Joyanes, L.; Rodríguez, L.; Fernández, J. (2000), Fundamentos de programación: libro de problemas, McGraw-Hill.
- [JOYA02] Joyanes, L.; Castillo, A.; Sánchez, L. & Zahonero, I. (2002), Programación en C. Libro de problemas, McGraw-Hill.
- [GARC02] García, F.; A. Calderón; Carretero, J.; Fernández, J. & Pérez, J. (2002), Problemas resueltos de programación C, Thompson.
- [GARC01] García, F.; Carretero, J.; Fernández, J. & Calderón, A. (2001), El lenguaje de programación C, Prentice-Hall.