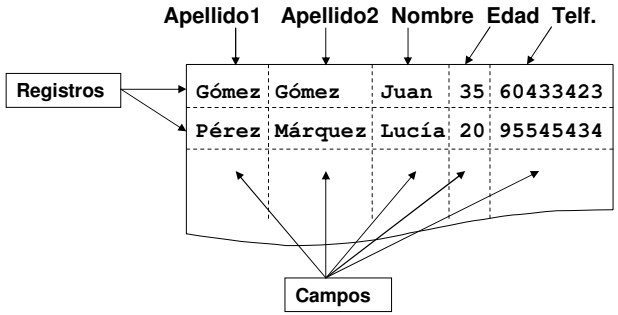
Ficheros
conceptos

- Contienen los datos que se almacenan en dispositivos de almacenamiento masivo: disquetes, discos duros, DROM, etc.
- Conjunto de datos relacionados entre sí.
- Organización de los datos:
  - Sin una estructura regular: texto, código C.
  - En registros.



Manejo de ficheros en C

- Los tipos de datos y funciones necesarias para el manejo de ficheros se encuentran en la librería de entrada/salida estándar `stdio` (standard input/output).
  - `#include <stdio.h>`
- Los ficheros se manejan utilizando variables de tipo **`FILE *`**
  - Ejemplos:

```
FILE *fichero;
FILE *nominas, *ventas;
```

El asterisco debe aparecer delante del nombre de la variable.

  - Estas variables pueden tener un valor **nulo** que se representa por la constante **`NULL`**.
- Los pasos generales para manejar ficheros son los siguientes:
  - Abrir el fichero.
  - Realizar las operaciones de entrada/salida (lectura y/o escritura de datos en el fichero).
  - Cerrar el fichero.

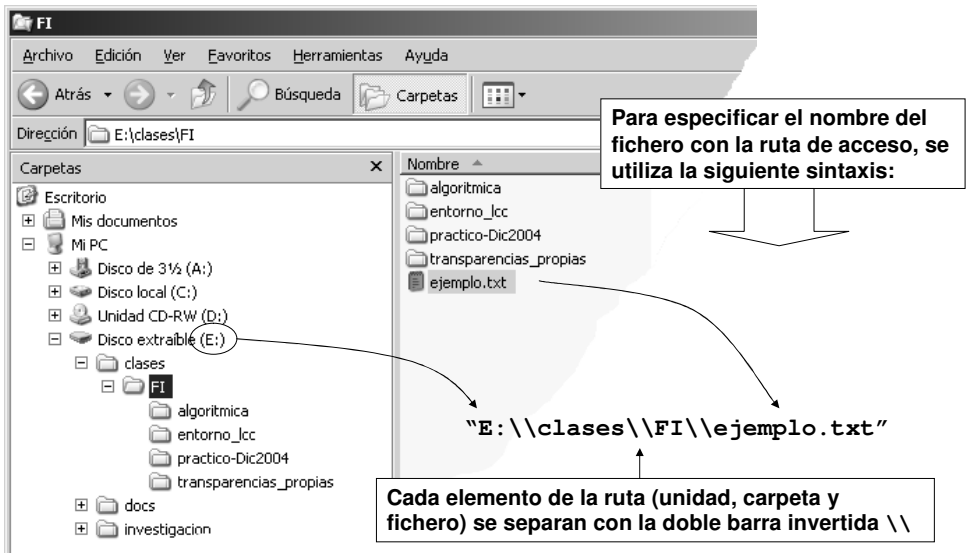
Apertura del fichero
Función `fopen`

- Mediante la apertura asociamos el fichero con una variable de tipo `FILE*`
- Una vez realizada la apertura, la variable se utiliza para acceder al fichero.
- Se utiliza la siguiente función:

```
FILE *fopen( char nombre[], char modo[] );
```

  - **Parámetros**:
    - **nombre**: Es una cadena de caracteres que contiene el nombre del fichero. Puede contener la ruta.
    - **modo**: Es una cadena de caracteres que indica el modo de apertura del fichero. Puede contener los siguientes valores:
      - **`"rt"`** El fichero se abre para leer.
      - **`"wt"`** El fichero se abre para escribir. Se borra y se escribe al principio.
      - **`"at"`** El fichero se abre para añadir. Se añaden los datos al final.
  - **Valor devuelto**:
    - Si el fichero no ha podido abrirse, devuelve `NULL`.
    - Si el fichero se ha abierto correctamente, devuelve un valor de tipo `FILE*` que se puede utilizar para acceder al fichero.

Apertura del fichero
Función `fopen`: nombre del fichero



Apertura del fichero
Función `fopen`: modo de apertura

- Existen tres modos de apertura:
  - Lectura (**`"rt"`**)
    - Este modo se utiliza para leer datos del fichero.
    - Si el fichero no existe, no se puede abrir y `fopen` devuelve `NULL`.
  - Escritura (**`"wt"`**)
    - Este modo se utiliza para escribir datos en el fichero desde el principio.
    - Si el fichero existe, se borra su contenido.
    - Si el fichero no existe, se crea un fichero nuevo vacío con el nombre indicado.
    - Deben existir todas las carpetas de la ruta especificada; si alguna no existe, `fopen` no podrá abrir el fichero.
  - Añadir (**`"at"`**)
    - Este modo se utiliza para escribir datos al final del fichero.
    - Si el fichero existe, se mantiene su contenido.
    - Si el fichero no existe, se crea un fichero nuevo vacío con el nombre indicado.
    - Deben existir todas las carpetas de la ruta especificada; si alguna no existe, `fopen` no podrá abrir el fichero.

Apertura del fichero
Función `fopen`: ejemplo 1a

```
#include <stdio.h>
void main() {
        FILE *fa, *fb;
        fa = fopen( "c:\\dir1\\prueba1.txt", "wt" );
        if (fa == NULL) {
                printf( "Error\n" );
                return;
        }

        fb = fopen( "c:\\dir1\\prueba2.txt", "wt" );
        if (fb == NULL) {
                printf( "Error\n" );
                return;
        }
        ...
}
```

Abrimos el fichero "`prueba1.txt`" para escritura. Si el fichero existe, se borra su contenido.

Comprobamos si el fichero se ha podido abrir.

El valor devuelto por `fopen` se guarda en `fa`. Ahora podemos utilizar `fa` para acceder al fichero "`prueba1.txt`".

Podemos abrir más de un fichero

Resto de sentencias del programa

Apertura del fichero
Función `fopen`: ejemplo 1b

El programa anterior se puede escribir de la siguiente manera:

```
#include <stdio.h>
void main() {
        FILE *fa, *fb;
        fa = fopen( "c:\\dir1\\prueba1.txt", "wt" );
        if (fa == NULL) {
                printf( "Error\n" );
        }else {
                fb = fopen( "c:\\dir1\\prueba2.txt", "wt" );
                if (fb == NULL) {
                        printf( "Error\n" );
                }else {
                        ...
                }
        }
}
```

Resto de sentencias del programa

Escritura de datos en el fichero
Función `fprintf`

- La función `fprintf` se utiliza igual que `printf`.
- Diferencias entre `fprintf` y `printf`:
  - La función `fprintf` tiene un primer parámetro de tipo `FILE*`
  - La función `fprintf` escribe en el fichero indicado por su primer parámetro (en vez de hacerlo por pantalla).

```
int fprintf( FILE *f, char texto_de_formato[], ... );
```

Contiene información sobre el fichero en el que se va a escribir.

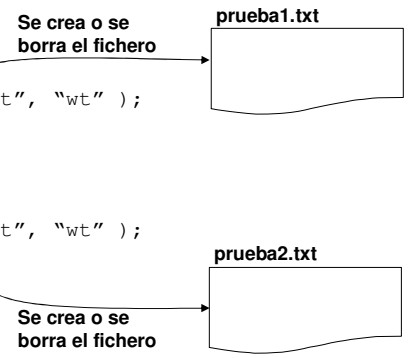
Contiene el texto que se va a escribir (junto con los códigos de formato: %d, %f, etc).

Valores que se van a imprimir dentro del texto.

- La función `fprintf` devuelve el nº de *bytes* o caracteres escritos en el fichero. En caso de error, devuelve un valor negativo.

Escritura de datos en el fichero
Función `fprintf`: ejemplo 1a

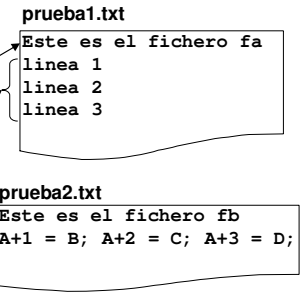
```
#include <stdio.h>
void main() {
        FILE *fa, *fb;
        int i;
        fa = fopen( "c:\\dir1\\prueba1.txt", "wt" );
        if (fa == NULL) {
                printf( "Error\n" );
                return;
        }
        fb = fopen( "c:\\dir1\\prueba2.txt", "wt" );
        if (fb == NULL) {
                printf( "Error\n" );
                return;
        }
```



Continúa...

Escritura de datos en el fichero
Función `fprintf`: ejemplo 1b

```
fprintf( fa, "Este es el fichero fa\n" );
fprintf( fb, "Este es el fichero fb\n" );
for (i=1; i<4; i++) {
        fprintf( fa, "linea %d\n", i );
        fprintf( fb, "A+%d = %c; ", i, 'A'+i );
}
...
```



Cierre del fichero
Función `fclose`

- Cuando un programa termina de utilizar un fichero, debe cerrarlo.
- La variable de tipo `FILE*` asociada al fichero queda libre y puede ser utilizada para acceder a otro fichero.
- Se utiliza la siguiente función:

```
int fclose( FILE *fich );
```

  - **Parámetros**:
    - **fich**: Valor de tipo `FILE*` asociado al fichero que queremos cerrar.
  - **Valor devuelto**:
    - Cero si se ha cerrado sin problemas.
    - El valor **`EOF`** si el fichero no se ha podido cerrar (`EOF` es una constante definida en la librería `stdio`).

Cierre del fichero
Función `fclose`: ejemplo

```
#include <stdio.h>
void main() {
        FILE *fa, *fb;
        int i;
        fa = fopen( "c:\\dir1\\prueba1.txt", "wt" );
        if (fa == NULL) {
                printf( "Error\n" );
                return;
        }
        fb = fopen( "c:\\dir1\\prueba2.txt", "wt" );
        if (fb == NULL) {
                printf( "Error\n" );
                return;
        }
        fprintf( fa, "Este es el fichero fa\n" );
        fprintf( fb, "Este es el fichero fb\n" );
        for (i=1; i<4; i++) {
                fprintf( fa, "linea %d\n", i );
                fprintf( fb, "A+%d = %c; ", i, 'A'+i );
        }
```

Apertura de ficheros

Operaciones de entrada/salida

Continúa...

Cierre del fichero
Función `fclose`: ejemplo

```
        fclose(fa);
        fclose(fb);

        fb = fopen( "c:\\tmp\\prueba3.txt", "rt" );
        ...
}
```

Cierre de los ficheros: el programa no va a seguir escribiendo en los ficheros.

Las variables `fa` y `fb` ya no están asociadas a ningún fichero.

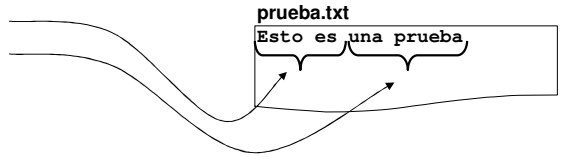
Podemos volver a utilizar `fb` para abrir un nuevo fichero y realizar operaciones de entrada/salida sobre él.

Funciones de escritura de ficheros
Función `fputs`

```
int fputs( char cad[], FILE *fich );
```

- Escribe el contenido de la cadena de caracteres **`cad`** en el fichero asociado a **`fich`**.
- Devuelve un valor >= 0 si la escritura se ha realizado sin problemas. En caso de error, devuelve el valor **`EOF`**.
- Ejemplo:

```
FILE *f = fopen( "c:\\tmp\\prueba.txt", "wt" );
char texto[] = "una prueba";
fputs( "Esto es ", f );
fputs( texto, f );
```

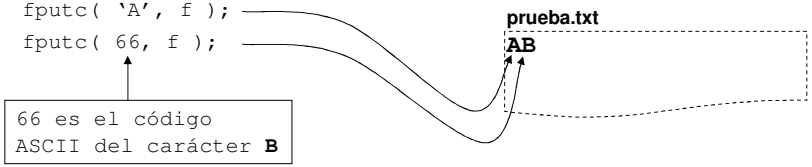


Funciones de escritura de ficheros
Función `fputc`

```
int fputc( int caracter, FILE *fich );
```

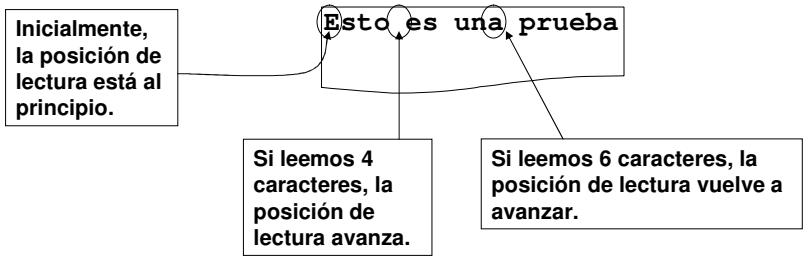
- Escribe en el fichero asociado a **`fich`** el carácter contenido en el parámetro **`caracter`**.
- Devuelve el carácter escrito si la escritura se ha realizado sin problemas. En caso de error, devuelve el valor **`EOF`**.
- Ejemplo:

```
FILE *f = fopen( "c:\\tmp\\prueba.txt", "wt" );
fputc( 'A', f );
fputc( 66, f );
```



Lectura de datos del fichero
Posición de lectura

- **Posición de lectura**: llamaremos posición de lectura al punto, dentro del fichero, donde se comenzará a leer los datos.
- Cuando se abre un fichero para leer (modo `"rt"`), la posición de lectura se sitúa al principio del fichero.
- Cada vez que se leen datos del fichero, la posición de lectura avanza para que la siguiente lectura comience en el lugar donde terminó la anterior.
- Ejemplo:



Lectura de datos del fichero
Función `fscanf`

- La función `fscanf` se utiliza igual que `scanf`.
- Diferencias entre `fscanf` y `scanf`:
  - La función `fscanf` tiene un primer parámetro de tipo `FILE*`
  - La función `fscanf` lee los datos del fichero indicado por su primer parámetro (en vez de leerlos del teclado).

```
int fscanf( FILE *f, char formato[], ... );
```

Contiene información sobre el fichero en el que se va a leer.	Contiene los códigos de formato de los valores que se van a leer (%d, %f, etc).	Variables en las que se van a guardar los datos leídos.

- La función `fscanf` devuelve el nº de valores que han sido leídos del fichero y almacenados en las variables. Si no ha podido leer más valores porque ha llegado al final del fichero, devuelve **`EOF`**.

Lectura de datos del fichero
Función `fscanf`: códigos de formato

- Se considera un separador a los siguientes caracteres:
  - Carácter espacio: `' '`
  - Salto de línea: `'\n'`
  - Tabulador: `'\t'`
- Lectura de valores numéricos: **%d**, **%u**, **%f**, **%lf**, etc
  - Se salta todos los separadores hasta encontrar el principio de un número: un dígito, `'+'`, `'-'` o `'.'` (para los tipos `float` y `double`)
  - Si antes de encontrar un número, encuentra un carácter distinto de los anteriores, la lectura termina y no se lee ningún valor numérico del fichero.
- Lectura de cadena de caracteres: **%s**
  - Se salta todos los separadores hasta encontrar un carácter distinto
  - Lee una cadena de caracteres delimitada por separadores
- Lectura de caracteres: **%c**
  - `fscanf` no se salta ningún carácter, lee el primer carácter que encuentre.

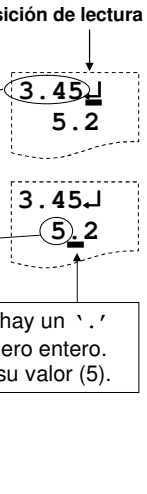
Lectura de datos del fichero
Función `fscanf`: ejemplo

```
#include <stdio.h>

void main() {
   float v1;
   int v2;
   char c1, c2, c3;
   FILE *f = fopen( "c:\\dir1\\ejemplo.txt", "rt" );
   if (f != NULL) {
      fscanf( f, "%f", &v1 );
      fscanf( f, "%d", &v2 );
      fscanf( f, "%d", &v2 );
      fscanf( f, "%c%c", &c1, &c2 );
   }
   fclose(f);

}
```

Apertura: `FILE *f = fopen( "c:\\dir1\\ejemplo.txt", "rt" );` / `if (f != NULL) {`

Procesado: `fscanf( f, "%f", &v1 );` / `fscanf( f, "%d", &v2 );` / `fscanf( f, "%d", &v2 );` / `fscanf( f, "%c%c", &c1, &c2 );` / `}`

Cierre: `fclose(f);`



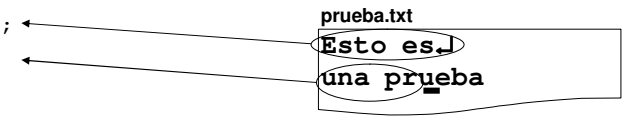
NOTA: el símbolo (↵) representa un salto de línea

Funciones de lectura de ficheros
Función `fgets`

```
char *fgets( char cadena[], int max, FILE *fich );
```

- Lee una secuencia de caracteres del fichero asociado al parámetro **`fich`** y los almacena en el parámetro **`cadena`**.
- La lectura termina cuando se hayan leído **`max-1`** caracteres, se encuentre un salto de línea o se acabe el fichero.
- Si se encuentra un salto de línea, este se incluirá en la cadena.
- Ejemplo:

```
FILE *f = fopen( "c:\\tmp\\prueba.txt", "rt" );
char cad[80];
fgets( cad, 80, f );
fgets( cad, 6, f );
```

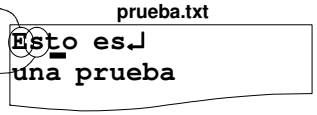


Funciones de lectura de ficheros
Función `fgetc`

```
int fgetc( FILE *fich );
```

- Lee un carácter del fichero asociado al parámetro **`fich`**.
- Devuelve el carácter leído. Si no se puede leer ningún carácter, devuelve el valor **`EOF`**.
- Ejemplo:

```
FILE *f = fopen( "c:\\tmp\\prueba.txt", "rt" );
char c1, c2;
c1 = fgetc( f );
c2 = fgetc( f );
```



Detección del fin del fichero
Función `feof`

- Cuando la posición de lectura del fichero se encuentra al final del fichero, no se puede seguir leyendo.
- Durante la lectura de datos de un fichero, el programa debe comprobar si se ha llegado al final del fichero.
- La siguiente función nos permite detectar si se ha llegado al final del fichero:

```
int feof( FILE *fich );
```

  - **Valor devuelto**:
    - VERDADERO (distinto de cero): Si en la última operación de lectura se ha intentado leer fuera del fichero; es decir, si se ha intentado leer después de llegar al final del fichero.
    - FALSO (cero): En caso contrario

Ficheros
Tratamiento secuencial

- La secuencia de pasos que seguiremos para el tratamiento secuencial de ficheros es la siguiente:

```
Abrir fichero
Leer un registro
MIENTRAS no sea fin de fichero HACER
   Procesar el registro leído
   Leer un registro
FIN MIENTRAS
Cerrar fichero
```

Ficheros
Tratamiento secuencial: ejemplo 1a

Realizar un programa que lea los registros de un fichero de temperaturas que contiene un entero por cada línea (cada línea corresponde a un registro). El programa debe imprimir por pantalla los valores leídos.

```
#include <stdio.h>
void main() {
   int valor;
   FILE *f = fopen( "c:\\ejemplo.txt", "rt" );
   if (f == NULL) {
      printf( "ERROR\n" );
      return;
   }
   fscanf( f, "%d", &valor );
   while (!feof(f)) {
      printf( "%d\n", valor );
      fscanf( f, "%d", &valor );
   }
   fclose(f);
}
```

Abrimos el fichero

Leemos un registro

Procesamos el registro

Ficheros
Tratamiento secuencial: ejemplo 1b

Si el último registro no termina con un salto de línea, el valor de dicho registro no aparece en pantalla (no es procesado).

```
#include <stdio.h>
void main() {
   int valor;
   FILE *f = fopen( "c:\\ejemplo.txt", "rt" );
   if (f == NULL) {
      printf( "ERROR\n" );
      return;
   }
   fscanf( f, "%d", &valor );
   while (!feof(f)) {
      printf( "%d\n", valor );
      fscanf( f, "%d", &valor );
   }
   fclose(f);
}
```

Supongamos que la posición de lectura se encuentra en la marca (▬) y que se va a ejecutar la siguiente línea

45↵
5↵
1▬0

Se saltan los separadores hasta encontrar el carácter '1'.

Se lee el carácter '0'.

Se intenta leer fuera del fichero.

Ficheros
Tratamiento secuencial: ejemplo 2a

Tenemos un fichero "temp.txt" con las temperaturas en ºC de tres sensores (llamados A, B y C). El siguiente programa copia las temperaturas del sensor A en un fichero llamado "temp_a.txt".

Cada línea del fichero corresponde a un registro que contiene el nombre del sensor (un carácter) y la temperatura (un entero) separados por un espacio.

Por ejemplo, para el siguiente fichero "temp.txt", debe generarse el fichero "temp_a.txt" que se muestra:

temp.txt

A 45↵
B 60↵
A 55↵
C 32↵

Temp_a.txt

45↵
55↵

Ficheros
Tratamiento secuencial: ejemplo 2b

```
#include <stdio.h>
void main() {
   FILE *ftemp, *fsal;
   char sensor, aux;
   int temp;
   ftemp = fopen( "c:\\dir\\temp.txt", "rt" );
   if (ftemp == NULL) {
      printf( "ERROR\n" );
      return;
   }
   fsal = fopen( "c:\\dir\\temp_a.txt", "wt" );
   if (fsal == NULL) {
      printf( "ERROR\n" );
      fclose( ftemp );
      return;
   }
```

Abrimos el fichero en modo lectura

Abrimos el fichero en modo escritura

Antes de terminar, debemos cerrar el fichero "temp.txt"

Continúa...

Ficheros
Tratamiento secuencial: ejemplo 2c

```
   fscanf( ftemp, "%c", &sensor );
   fscanf( ftemp, "%d", &temp );
   fscanf( ftemp, "%c", &aux );
   while (!feof(ftemp)) {
      if (sensor == 'a') {
         fprintf( fsal, "%d\n", temp );
      }
      fscanf( ftemp, "%c", &sensor );
      fscanf( ftemp, "%d", &temp );
      fscanf( ftemp, "%c", &aux );
   }
   fclose( ftemp );
   fclose( fsal );
}
```

Leemos un registro del fichero "temp.txt"

Procesamos el registro leido y escribimos un registro en el fichero "temp_a.txt"

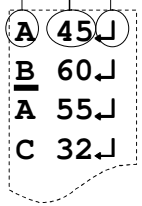
Leemos un registro del fichero "temp.txt"

Cerramos todos los ficheros

Ficheros
Tratamiento secuencial: ejemplo 2d

Lectura de un registro del fichero “temp.txt”:

```
fscanf( ftemp, "%c", &sensor );
fscanf( ftemp, "%d", &temp );
fscanf( ftemp, "%c", &aux );
```



La variable **aux** se utiliza para leer el salto de línea del registro y conseguir que la posición de lectura se sitúe al principio del siguiente registro