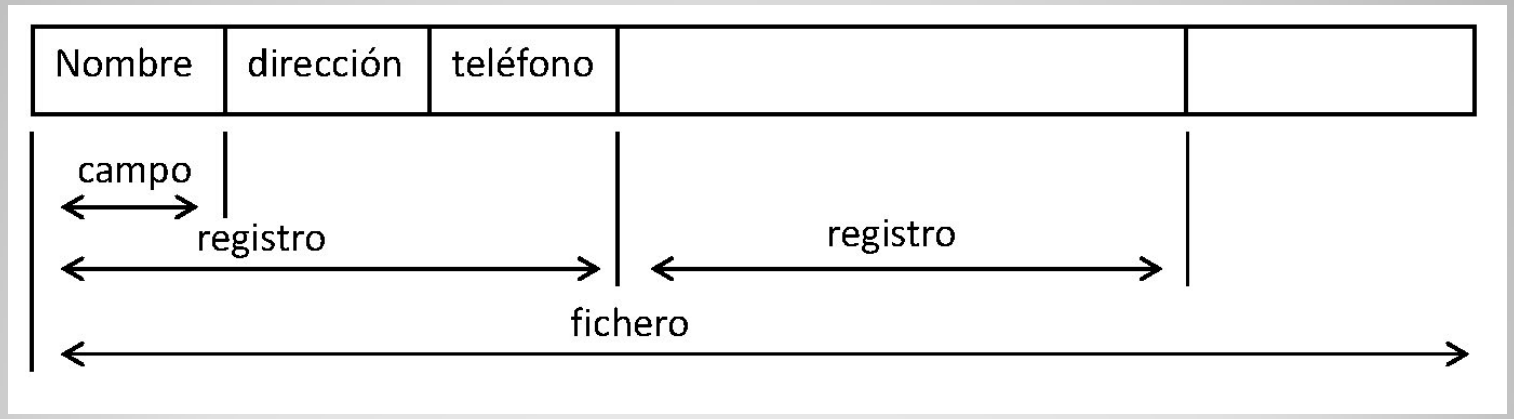
Curso C

Ficheros

Ficheros

- Un fichero o archivo es una colección de información almacenada en un soporte para poder manipularla en cualquier momento. La información se almacena como un conjunto de registros. (registro más simple es un char)



- Para obtener independencia del dispositivo, las funciones de C no trabajan directamente con el fichero, sino con un intermediario (flujo o stream)

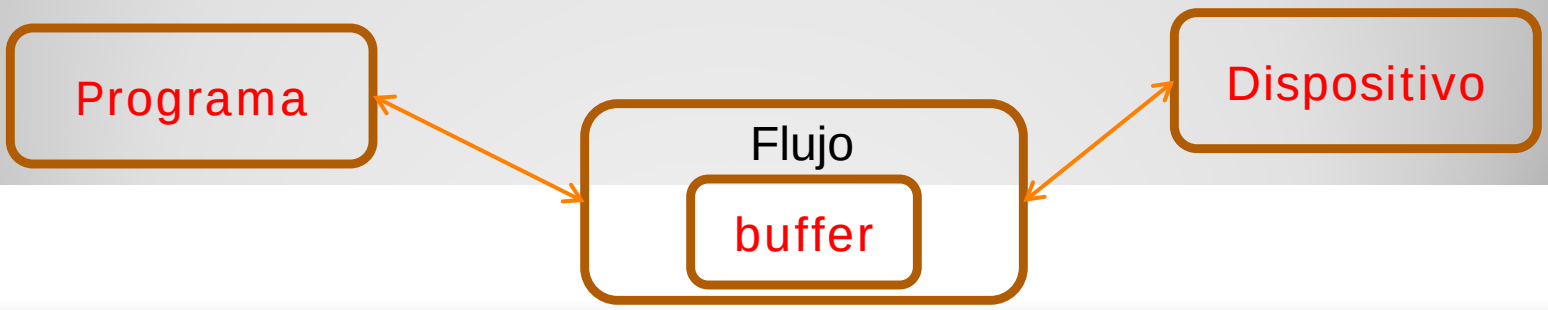
Flujo o stream

- Los algoritmos para leer o escribir son siempre parecidos

Leer	Escribir

Abrir un flujo desde un fichero
Mientras haya información
  Leer
Cerrar el Flujo
	Abrir un flujo hacia un fichero
	Mientras haya información
	  Escribir
	Cerrar el Flujo

- C usa una estructura llamada FILE, el cual, entre otros contiene un buffer



Trabajar con ficheros

- Abrir fichero
  - FILE* fopen(const char* nombre, const char* modo);

Modo	Descripción
“r”	Abre para leer. Si no existe ERROR
“w”	Abre para escribir. Si no existe se crea. Si existe se vacía.
“a”	Abre para añadir al final. Si no existe se crea
“r+”	Abre para leer y escribir. Fichero debe existir
“w+”	Abre escribir y leer. Si no existe se crea. Si existe se vacía.
“a+”	Abre para leer y añadir. Si no existe se crea

- Al abrir el puntero está colocado al principio del fichero, excepto para modo añadir que se coloca al final.

Trabajar con ficheros

- Abrir fichero

```
main(){
 FILE* pf;
 pf = fopen("datos","w");
 if (pf==NULL) printf("ERROR");
 ...
 fclose(pf);
}
```

- Cerrar un Fichero
  - int fclose(FILE* pf); -> si ok, devuelve 0, sino EOF
- Para saber cuando hemos llegado al final del fichero usaremos:
  - int feof(FILE* pf) -> devuelve valor != de 0 si intentamos leer más allá del final. En cualquier otro caso devuelve 0

Gestión errores ficheros

- Cuando hacemos una operación con un fichero y da error, podemos detectarlo preguntando por:
  - int ferror(FILE *pf) -> Devuelve != 0 si ha habido error
- Debemos limpiar el indicador de error usando:
  - void clearerr(FILE *pf)

Trabajar con ficheros: Entrada/Salida

- E/S carácter a carácter
  - int fputc(int car, FILE* pf) esta función escribe el carácter car en el fichero pf (donde apunte). Devuelve el carácter escrito o EOF si hay error.
  - int fgetc(FILE *pf) esta función lee un carácter desde el fichero. Devuelve EOF si error

Realizar un programa que copie un fichero cualquiera en otro. Ej. Copiar fichero1 fichero2

- E/S cadenas de caracteres
  - int fputs(const char* cad, FILE *pf) esta función copia la cadena de caracteres cad en el fichero asociado a pf. El \0 no se copia. Devuelve EOF si error
  - char *fgets(char *cad, int n, FILE*pf) Lee cadena de caracteres y almacena en cad. Lee hasta encontrar \n o fin de fichero o hasta n-1 caracteres. Añade el \0. Devuelve puntero al comienzo de cad.

Trabajar con ficheros

- Posicion de un puntero de Lectura/Escritura
  - Cuando hacemos lecturas o escrituras en ficheros, el puntero de la posición en el fichero se actualiza automáticamente.
  - Usando ftell, obtenemos la posición relativa al inicio del fichero
    - long ftell(FILE *pf)
  - También podemos posicionarnos donde deseemos en el fichero usando:
    - int fseek(FILE*pf, long desp, int pos);
    - Donde desp es el numero de bytes a desplazarse desde la posición pos.

pos	Significado
SEEK_SET	Inicio del fichero
SEEK_CUR	Posición actual
SEEK_END	Ultima posición del fichero

Trabajar con ficheros: Entrada/Salida con formato

- int fprintf(FILE* pf, const char *formato[,arg]…) esta función escribe argumentos arg con el formato especificado en el fichero pf (donde apunte). Devuelve el número de caracteres escritos o valor negativo si error.
- int fscanf(FILE *pf, const char* formato[,arg]…) esta función lee argumentos arg con el formato especificado. Devuelve EOF si fin fichero, 0 si no lee nada o numero de datos leidos

Trabajar con ficheros: E/S con registros

- size_t fwrite(const void*buffer, size_t n, size_t c, FILE* pf) esta función permite escribir c elementos de longitud n bytes (c x n bytes) almacenados en buffer en fichero asociado a pf. Devuelve el numero de elementos escritos.
- size_t fread(void *buffer, size_t n, size_t c, FILE* pf) permite leer c elementos de tamaño n bytes (c x n bytes) y almacenarlos en buffer desde el fichero asociado a pf. Devuelve numero de elementos leidos. Si es menor que c entonces error o fin de fichero

Datos a Impresora

```
FILE* pfs=NULL;
if((pfs=fopen("lpt1","w"))==NULL)
    printf("ERROR, impresora no disponible");
else{
  fprintf(pfs,"Esta cadena se imprime en
   impresora");
  fprintf(pfs, "\f"); //Avance pagina
}
```

En Unix/Linux usar "/dev/lp0"

- int fflush(FILE*pf) fuerza a volcar el contenido del buffer asociado a pf al fichero.

Acceso Aleatorio en ficheros

. Usando ftell, fseek, fread, fwrite

Ejemplo Pizarra

- Archivos de TEXTO frente a BINARIOS
  - Todos los modos vistos hasta ahora leen y escriben en archivos de TEXTO. Es decir, se almacenan como cadenas de caracteres.
  - POCO eficiente.
  - Modo BINARIO almacena en formato binario, al igual que en memoria. Ej. Un int (usa 4 bytes), etc.
  - Para abrir en modo binario añadir la letra “b” al modo de apertura.