



Informática

Ingeniería en Tecnologías Industriales

Entrada y salida por ficheros en lenguaje C

V1.3

© Autores



Entrada y salida por ficheros lenguaje C

- Ficheros y flujos en lenguaje C
- Apertura y cierre de un fichero
 - Fin de fichero
- Entrada y salida de texto
 - Entrada y salida de caracteres
 - Entrada y salida de cadenas
- Lectura y escritura de datos binarios
- Entrada y salida con formato
- Entrada y salida mediante acceso directo
- Otras operaciones sobre ficheros
 - Función `ftell()`
 - Función `rewind()`
 - Función `remove()`
 - Función `fflush()`
 - Función `tmpfile()`

V1.3

2

Ficheros y flujos en lenguaje C (I)

- El lenguaje C proporciona mecanismos para la entrada y salida por ficheros
 - No dependen del dispositivo físico sobre el que actúen
 - Son funciones genéricas, potentes y flexibles
 - Distingue dos tipos de elementos
 - *Ficheros*
 - *Flujos o streams*
- **Fichero**: Dispositivo físico y real sobre el que se realizan las operaciones de entrada o salida
- **Flujo (stream)**: Ente abstracto sobre el que se realiza una operación de entrada o salida genérica
 - Los flujos pueden asociarse a dispositivos físicos o a ficheros:
 - El *flujo de salida estándar* se asocia a la pantalla
 - El *flujo de entrada estándar* se asocia al teclado

V1.3

© Autores

3

Ficheros y flujos en lenguaje C (II)

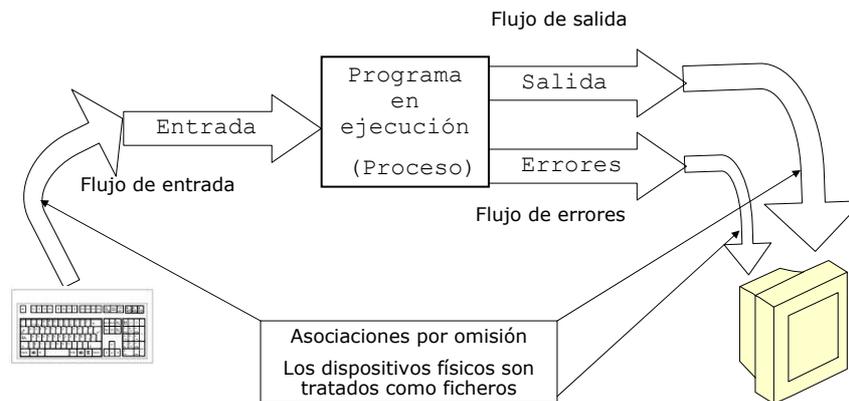
- El concepto de *fichero* puede aplicarse, además de a ficheros en disco, a impresoras, terminales, controladores, etc.
- El concepto de *flujo* puede entenderse como un *buffer* de datos o un dispositivo lógico que proporciona un camino para los datos en las operaciones de entrada o salida
- Cuando se abre un fichero para operaciones de entrada y/o salida, se le asocia un flujo, un camino que permitirá la transferencia de información entre ese fichero y el programa.
- El lenguaje C contempla los ficheros como flujos continuos de bytes que pueden ser accedidos de forma individual:
 - Todas las funciones de E/S sirven para todos los flujos: solo hay que redirigirlos al fichero deseado

V1.3

© Autores

4

Ficheros y flujos en lenguaje C (III)



V1.3

© Autores

5

Ficheros y flujos en lenguaje C (IV)

- El sistema operativo (DOS/Windows) admite que un fichero pueda ser abierto de dos modos:
 - En *modo texto*: Los bytes del flujo son códigos ASCII
 - En *modo binario*: Los bytes del flujo son códigos binarios de 8 bits
- En los sistemas Unix y Linux no existe tal distinción
 - No hay ninguna indicación añadida al modo de apertura de un fichero

V1.3

© Autores

6

Ficheros y flujos en lenguaje C (V)

- En *modo texto* el final de la línea se representa
 - En lenguaje C: como `'\n'` (En ASCII CR: Retorno de carro, código 10)
 - En el sistema operativo Windows: como la sucesión de los códigos ASCII: CR+LF (Retorno de carro+Salto de línea, códigos 10 y 13)
 - Las funciones de lectura en lenguaje C al encontrar la secuencia CR+LF, la convierten a `'\n'`
 - Las funciones de lectura al encontrar el código correspondiente a Ctrl-D (Ctrl-Z en Windows) lo interpretan como EOF (*End of file*, fin de fichero).

V1.3

© Autores

7

Ficheros y flujos en lenguaje C (VI)

- Ejemplo: Si en un fichero de texto del DOS (*.TXT) escribimos
En un lugar de la Mancha,
de cuyo nombre no quiero acordarme.
- Al leer el fichero en modo binario desde un programa en lenguaje C se obtendrá la siguiente secuencia
En un lugar de la Mancha,\r\nde cuyo nombre no quiero acordarme.EOF
- Si se abre en modo texto, lo que se recogerá será lo siguiente:
En un lugar de la Mancha,\nde cuyo nombre no quiero acordarme.EOF

V1.3

© Autores

8

Ficheros y flujos en lenguaje C (VII)

- Al ejecutar un programa en modo consola, automáticamente el sistema operativo abre tres flujos estándar:
 - `stdin`
 - Entrada estándar. Se asocia al teclado
 - `stdout`
 - Salida estándar. Se asocia a la pantalla
 - `stderr`
 - Salida estándar de errores. Se asocia a la pantalla
- Los flujos estándar tienen asociado un buffer de memoria física cada uno
- Las E/S por consola operan igual que las E/S por ficheros, pero redirigen la operación real sobre los flujos estándar `stdin`, `stdout` y `stderr`

V1.3

© Autores

9

Apertura y cierre de un fichero (I)

- En lenguaje C, para acceder a los ficheros se precisa un **descriptor del fichero** (puntero de tipo `FILE`)
 - Declaración:

```
FILE *puntero;
```
 - `FILE` es una constante definida en `stdio.h`
 - El descriptor apunta a un buffer que contiene toda la información necesaria para operar con el fichero
 - Se utiliza para hacer referencia al fichero en todas las operaciones sobre este
 - Debe declararse antes de utilizarse
 - El descriptor se inicializa al abrir sin error un fichero al que se le hace apuntar

V1.3

© Autores

10

Apertura y cierre de un fichero (II)

- Antes de cualquier operación sobre un fichero es preciso **abrir el fichero**, utilizando la función `fopen()`

- Declaración:

```
FILE *fopen(char *nombrearchivo,
            char *modo);
```

- Devuelve

- Un descriptor de tipo `FILE` que apunta al fichero abierto
- Un descriptor nulo (`NULL`) en caso de error

- Recibe dos cadenas de caracteres:

- La primera con el nombre del fichero (con la ruta de acceso)
- La segunda con el modo de apertura que se selecciona

V1.3

© Autores

11

Apertura y cierre de un fichero (III)

MODOS DE APERTURA DE LOS FICHEROS EN LENGUAJE C

Modo de apertura	Cadena de modo de apertura		Observaciones
	Modo texto	Modo binario	
Abrir para leer	"r"	"rb"	Si no existe, se produce error
Crear para escribir	"w"	"wb"	Si existe, se pierde el contenido
Abrir o crear para añadir	"a"	"ab"	Si no existe, se crea
Abrir para leer y/o escribir	"r+"	"rb+"	Debe existir
Crear para leer y/o escribir	"w+"	"wb+"	Si existe, se pierde el contenido
Abrir o crear para añadir y/o leer	"a+"	"ab+"	Si no existe, se crea

V1.3

© Autores

12

Apertura y cierre de un fichero (IV)

- Mientras el fichero esté abierto, el descriptor correspondiente apunta a una estructura que contiene toda la información necesaria sobre el fichero: Nombre, tamaño, atributos, posición del apuntador para lecturas y/o escritura, etc.
- Al finalizar el programa, si termina normalmente, los ficheros que estuviesen abiertos son cerrados por el sistema operativo.
- Un fichero no cerrado correctamente queda inutilizado y la información que pudiera contener se pierde y queda inaccesible
- En previsión de finalizaciones anómalas, tras las operaciones de lectura y/o escritura, es preciso cerrar los ficheros, utilizando la función `fclose()`:
 - Declaración

```
int fclose(FILE *descriptor);
```
 - Devuelve
 - un entero de valor cero si el cierre ha sido correcto
 - EOF en caso de error
 - Recibe como argumento el descriptor al fichero que se quiere cerrar

V1.3

© Autores

13

Apertura y cierre de un fichero (V)

- Ejemplo de apertura y cierre de un fichero:

```
FILE *pf; /* descriptor a fichero */
if ((pf=fopen("misdatos/prueba.x","w+")) == NULL)
{
    puts("\nNo es posible crear el fichero");
    exit(0);
}
else printf("\nEl fichero se ha abierto");
/* Tras realizar todas las operaciones necesarias,
es preciso cerrar el fichero antes de finalizar
el programa */
fclose(pf);
/* El fichero ha quedado cerrado */
```

V1.3

© Autores

14

Apertura y cierre de un fichero (VI)

- El **carácter de fin de fichero** está representado por la constante simbólica `EOF`, definida en `stdio.h`
 - Es el último byte del fichero
 - Cuando se leen bytes del fichero (con `fgetc()`) puede leerse el `EOF` y no distinguirse como último carácter, especialmente cuando se trata de flujos binarios
 - La función `feof()`, declarada en `stdio.h` devuelve un valor distinto de cero (verdadero/true) cuando se lee el byte de fin de fichero (`EOF`)

```
while (!feof(puntfile))
{
    /* Operaciones sobre el fichero abierto */
}
```

V1.3

© Autores

15

Entrada y salida de texto (I)

- Las funciones de lectura y escritura de caracteres en ficheros están definidas en `stdio.h`

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

- Lee un carácter en el fichero cuyo descriptor recibe
- Devuelve el carácter leído en un entero o `EOF` en caso de error

```
int fputc( int car, FILE *pf);
```

- Escribe un carácter en el fichero cuyo descriptor recibe
- Devuelve `EOF` en caso de error
- Recibe como argumentos:
 - `car`: El carácter a escribir
 - `pf`: El descriptor al fichero

V1.3

© Autores

16

Entrada y salida de texto (II)

- Ejemplos de lectura y escritura de un carácter en un fichero:

```
FILE *pf1, *pf2;
char letra;
pf1 = fopen("leer.txt", "r");
letra = fgetc(pf1); /* Lee un carácter */
pf2 = fopen("escribir.txt", "w");
fputc(letra, pf2); /* Escribe un carácter */
fclose(pf1);
fclose(pf2);
```

V1.3

© Autores

17

Entrada y salida de texto (III)

- Las funciones de lectura y escritura de cadenas de texto en ficheros están definidas en `stdio.h`

```
char * fgets(char *cad, int numcar, FILE *pf);
```

- Devuelve un puntero a la cadena leída o un puntero nulo en caso de error
- Recibe como argumentos
 - `cad`: Un puntero a la zona de memoria donde se almacenará la cadena
 - `numcar-1`: Es el número de caracteres a leer. Será añadido el carácter nulo
 - `pf`: El puntero al fichero
- Si encuentra un carácter de fin de línea (`\n`), éste será el último carácter leído

```
int fputs(char *puncad, FILE *pf);
```

- Devuelve en un entero el último carácter escrito o `EOF` en caso de error
- Recibe como argumentos
 - `puncad`: Un puntero a cadena que se quiere escribir
 - `pf`: El puntero al fichero

V1.3

© Autores

18

Entrada y salida de texto (IV)

- Ejemplos de lectura y escritura de una cadena de caracteres en un fichero:

```
FILE *pf1, *pf2;
char lect[50];
char escr[]="Mensaje a guardar en el fichero";
int num=50;
pf1 = fopen("leer.txt", "r");
fgets(lect, num, pf1);
/* Lee una cadena de, como máximo, 49 caracteres de
leer.txt */
pf2 = fopen ("escribir.txt", "w");
fputs(escr, pf2);
/* Escribe la cadena "Mensaje a guardar en el
fichero" en escribir.txt */
fclose(pf1);
fclose(pf2);
```

V1.3

© Autores

19

Lectura y escritura de datos binarios (I)

- Para la lectura de un conjunto de datos
`unsigned fread(void *buf, unsigned numbytes,
unsigned numdat, FILE *pf);`
- Para la escritura de un conjunto de datos
`unsigned fwrite(void *buf, unsigned numbytes,
unsigned numdat, FILE *pf);`
 - Devuelven el número de datos leídos o escritos
 - Reciben
 - `buf`: Un puntero a los datos que son leídos o escritos
 - `numbytes`: representa el número de bytes de que consta cada uno de los datos a a leer o escribir (se obtiene con `sizeof`)
 - `numdat`: representa el número total de datos, items o elementos a leer o escribir
 - `pf`: es el descriptor al fichero sobre el que van a leer o escribir

V1.3

© Autores

20

Lectura y escritura de datos binarios (II)

- Mediante el manejo de datos binarios el contenido de los ficheros puede ser similar al contenido de las variables en memoria
 - Es importante cuando se trabaja con estructuras y uniones
- Las funciones `fread` y `fwrite` pertenecen al ANSI C y están definidas en `stdio.h`
- Ejemplo de utilización:

```
FILE *pf;
float valor1=3.5, valor2;
pf=fopen ("archivo.dat", "ab+"); /* "a+" en Unix */
fwrite(&valor1, sizeof(valor), 1, pf); /*Escribe*/
fread(&valor2, sizeof(float), 1, pf); /* Lee */
```

V1.3

© Autores

21

Entrada y salida por ficheros con formato (I)

- Las funciones `fprintf()` y `fscanf()` son idénticas a `printf()` y `scanf()` respectivamente pero operan sobre un descriptor. También están declaradas en `stdio.h`

```
int fprintf(FILE *pf, const char*cadcontrol,
            listargumentos);
int fscanf(FILE *pf, const char*cadcontrol,
            listargumentos);
```

 - Argumentos que reciben
 - `pf`: Representa un puntero al fichero sobre el que opera la función
 - `cadcontrol`: Es la cadena en la que se incluyen los especificadores de formato y sus modificadores
 - `listargumentos`: Representa la lista de argumentos que se corresponden con los especificadores de formato de la cadena de control (lista de variables cuyos contenidos quieren escribirse o leerse sobre el fichero)
 - Devuelven
 - `fprintf()` devuelve en un entero el número de bytes escritos
 - `fscanf()` devuelve en un entero el número de campos correctamente escaneados y almacenados o EOF en caso de error.
 - La llamada `fprintf(stdout, "Número: %d", num);` es equivalente a la llamada `printf("Número: %d", num);`

V1.3

© Autores

22

Entrada y salida por ficheros con formato (II)

- Cuando se escribe en un fichero mediante la función `fprintf(pf, ...)` el contenido del fichero es similar al mostrado en pantalla con la función `fprintf(stdout, ...)`

- Ejemplo: Escritura y lectura con formato

```
FILE *pf;
int i = 100;
char c = 'C';
float f = 1.234;
pf = fopen("prueba.dat", "w+");
fprintf(pf, "%d %c %f", i, c, f);
/* Escribe en el fichero */
fscanf(pf, "%d %c %f", &i, &c, &f);
/* Lee los mismos datos en el fichero */
fclose(pf);
```

V1.3

© Autores

23

Entrada y salida mediante acceso directo (I)

- El acceso a los ficheros puede hacerse
 - En **modo secuencial**: los datos son leídos o escritos seguidos, sin saltos
 - En **modo aleatorio**: es posible acceder a cualquier posición para realizar una operación de lectura o de escritura
- Los punteros de tipo `FILE` apuntan a una estructura creada por el sistema operativo que controla las operaciones sobre ese fichero.
 - La estructura incluye un puntero de lectura-escritura que determina la posición actual en la que se va a escribir o a leer en cada momento
 - Al abrir un fichero, el puntero de lectura-escritura apunta al comienzo del fichero (salvo se si abre para añadir)
- La función `fseek()`, definida en `stdio.h`, permite el movimiento aleatorio por el fichero abierto estableciendo una nueva posición para el apuntador de lectura-escritura

V1.3

© Autores

24

Entrada y salida mediante acceso directo (II)

```
int fseek(FILE *pf, long nbytes, int origen);
```

- Devuelve un valor verdadero si el movimiento se realizó con éxito o cero en caso contrario
- Recibe
 - `pf`: Descriptor con el que se opera
 - `nbytes`: Número de bytes que queremos desplazar el apuntador de lectura-escritura del fichero con relación al punto de partida
 - `origen`: Representa el punto que se toma como origen o referencia. Se utilizan constantes simbólicas definidas en `stdio.h`:
 - `SEEK_SET` corresponde al principio del fichero
 - `SEEK_CUR` corresponde a la posición actual
 - `SEEK_END` corresponde al final del fichero

V1.3

© Autores

25

Entrada y salida mediante acceso directo (III)

- Ejemplo:

```
#define N 5 /* Será la fila 5 */
int matriz[20][4], *punt; /* 20 filas y 4 columnas */
FILE *pf;
punt = matriz;
.../* Apertura del fichero sin errores */
fwrite(punt, sizeof(int), 20*4, pf);
/* Escribe la matriz en un fichero */
fseek(pf, sizeof(int)*4*N, SEEK_SET);
/* Apunta a los datos de la fila N, al ser
4*sizeof(int) el tamaño de una fila */
fread(&matriz[N][0], sizeof(int), 4, pf);
/* Lee los datos de la fila N */
```

V1.3

© Autores

26

Otras operaciones sobre ficheros (I)

● Función `ftell()`

```
long ftell(FILE *pf);
```

- Devuelve un entero largo con la posición del apuntador de lectura-escritura del fichero con respecto al principio del fichero.
- Recibe el puntero a un fichero abierto
- Está definida en `stdio.h`

V1.3

© Autores

27

Otras operaciones sobre ficheros (II)

● Función `rewind()`

```
void rewind(FILE *pf);
```

- Inicializa el indicador de posición o apuntador de lectura-escritura haciendo que apunte al principio del fichero
- No devuelve nada
- Recibe el puntero a un fichero abierto
- Está definida en `stdio.h`

V1.3

© Autores

28

Otras operaciones sobre ficheros (III)

● Función `remove()`

```
int remove(char *nombrearchivo);
```

- Borra el fichero cuyo nombre se especifique en la cadena que recibe como argumento
- Devuelve cero si la operación tuvo éxito y -1 si se produce algún error
 - En caso de error, la variable global `errno`, definida en `errno.h` indicará el tipo de error
- Está definida en `stdio.h`

V1.3

© Autores

29

Otras operaciones sobre ficheros (IV)

● Función `tmpfile()`

```
FILE *tmpfile(void);
```

- Crea un fichero temporal que es borrado automáticamente cuando el fichero es cerrado o al terminar el programa
- El fichero temporal se crea en modo `"w+"`
- Devuelve un descriptor al fichero temporal creado o un puntero nulo si no se puede crear
- Está definida en `stdio.h`

V1.3

© Autores

30