



Tema 5: Entrada/Salida por archivos

Introducción a la programación II

Marcos Novalbos
Maximiliano Fernández
Alfonso Castro

Índice

1. Introducción
2. Flujos (streams)
3. Apertura y cierre de un archivo
4. Funciones de lectura y escritura
5. Archivos binarios en C
6. Acceso directo a los datos

5.1 Introducción

- Los datos que hemos tratado hasta el momento han residido en la memoria principal. Sin embargo, las grandes cantidades de datos se almacenan normalmente en un dispositivo de memoria secundaria. Estas colecciones de datos se conocen como **archivos o ficheros**.
- Hasta el momento se han realizado operaciones de entrada y salida:
 - Entrada desde el teclado por el usuario. La operación de entrada se denomina lectura
 - Salida al monitor. La operación de salida se denomina escritura
- En un lenguaje de programación, **un archivo o fichero está asociado al concepto lógico de flujo (stream)** que puede aplicarse a muchas cosas desde archivos de disco hasta terminales o una impresora.
- Una vez que el archivo está abierto, la información puede ser intercambiada entre este y el programa.

5.1 Introducción

- Las funciones de entrada/salida no están definidas en el lenguaje C, sino que están incorporadas en cada compilador de C bajo la forma de biblioteca de ejecución.
- La biblioteca `stdio.h` proporciona tipos de datos, macros y funciones para acceder a los archivos.
- En C, todos los ficheros almacenan bytes, y es al realizar la apertura y la escritura cuando se decide cómo y qué se almacena en el mismo; durante la declaración del fichero no se hace ninguna distinción sobre el tipo del mismo

5.1 Introducción

- Trataremos dos tipos de archivos, archivos de texto y archivos binarios. La operación de apertura se puede decidir según el tipo de archivo.
 - **Un archivo binario**
 - Es una secuencia de bytes que tienen una correspondencia uno a uno con un dispositivo externo.
 - Así que no tendrá lugar ninguna traducción de caracteres.
 - Además, el número de bytes escritos (leídos) será el mismo que los encontrados en el dispositivo externo.
 - Ejemplos de estos archivos son Fotografías, imágenes, texto con formatos, archivos ejecutables (aplicaciones), etc.
 - **Un archivo de texto**
 - Es una secuencia de caracteres organizadas en líneas terminadas por un carácter de nueva línea.
 - En estos archivos se pueden almacenar listas de canciones, descripción de fuentes de programas, base de datos simples, etc.
 - Los archivos de texto se caracterizan por ser "texto plano" (sin formato). Es decir, todas las letras tienen el mismo formato y no hay palabras subrayadas, en negrita, o letras de distinto tamaño o ancho.

5.1 Introducción

- El manejo de archivos en C se hace mediante el concepto de flujo (stream).
- Los flujos pueden estar abiertos o cerrados, conducen los datos entre el programa y los dispositivos externos.
- Con las funciones proporcionadas por `stdio.h` se pueden tratar:
 - Archivos secuenciales
 - De acceso directo
 - Archivos indexados ...
- Se ofrece la interfaz/estructura "FILE", pero no se deben acceder a sus parámetros directamente, no es portable entre compiladores ni versiones de C.
- En las siguientes transparencias se visualizan la estructura FILE en una distribución de c real.

5.1 Introducción

- MinGW/Cygwin provee herramientas de programación Open Source compatibles con aplicaciones MS-Windows, y que no dependen de DLLs externas.

```
/*  
 * The structure underlying the FILE type.  
 *  
 * Some believe that nobody in their right mind should make use of the  
 * internals of this structure. Provided by Pedro A. Aranda Gutierrez  
 * <paag@tid.es>.  
 */  
typedef struct _iobuf  
{  
    char*    _ptr;  
    int      _cnt;  
    char*    _base;  
    int      _flag;  
    int      _file;  
    int      _charbuf;  
    int      _bufsiz;  
    char*    _tmpfname;  
} FILE;
```

5.1 Introducción

- Con gcc en Linux se usa glibc y está definido en libio.h

```
struct _IO_FILE {
    int _flags;          /* High-order word is _IO_MAGIC; rest is flags. */
#define _IO_file_flags _flags

    /* The following pointers correspond to the C++ streambuf protocol. */
    /* Note: Tk uses the _IO_read_ptr and _IO_read_end fields directly. */
    char* _IO_read_ptr;  /* Current read pointer */
    char* _IO_read_end;  /* End of get area. */
    char* _IO_read_base; /* Start of putback+get area. */
    char* _IO_write_base; /* Start of put area. */
    char* _IO_write_ptr; /* Current put pointer. */
    char* _IO_write_end; /* End of put area. */
    char* _IO_buf_base;  /* Start of reserve area. */
    char* _IO_buf_end;   /* End of reserve area. */
    /* The following fields are used to support backing up and undo. */
    char *_IO_save_base; /* Pointer to start of non-current get area. */
    char *_IO_backup_base; /* Pointer to first valid character of backup area */
    char *_IO_save_end; /* Pointer to end of non-current get area. */
}
```


5.1 Introducción

- Con gcc en Linux se usa glibc y está definido en libio.h (continuación 1)

```
struct _IO_marker *_markers;

struct _IO_FILE *_chain;

int _fileno;
#if 0
int _blksize;
#else
int _flags2;
#endif
_IO_off_t _old_offset; /* This used to be _offset but it's too small. */

#define __HAVE_COLUMN /* temporary */
/* 1+column number of pbase(); 0 is unknown. */
unsigned short _cur_column;
signed char _vtable_offset;
char _shortbuf[1];

/* char* _save_gptr; char* _save_egptr; */

_IO_lock_t *_lock;
#ifdef _IO_USE_OLD_IO_FILE
};
```

5.1 Introducción

- Con gcc en Linux se usa glibc y está definido en libio.h (continuación 2)

```
struct _IO_FILE_complete
{
    struct _IO_FILE _file;
#ifdef
#if defined _G_IO_IO_FILE_VERSION && _G_IO_IO_FILE_VERSION == 0x20001
    _IO_off64_t _offset;
# if defined _LIBC || defined _GLIBCPP_USE_WCHAR_T
    /* Wide character stream stuff.  */
    struct _IO_codecvt *_codecvt;
    struct _IO_wide_data *_wide_data;
    struct _IO_FILE *_freeres_list;
    void *_freeres_buf;
# else
    void *__pad1;
    void *__pad2;
    void *__pad3;
    void *__pad4;
# endif
    size_t __pad5;
    int _mode;
    /* Make sure we don't get into trouble again.  */
    char _unused2[15 * sizeof (int) - 4 * sizeof (void *) - sizeof (size_t)];
#endif
};
```

5.2 Flujos (streams)

- **Un flujo (stream) es una abstracción que se refiere a un flujo o corriente de datos que fluyen entre un origen o fuente (productor) y un destino o sumidero (consumidor)**
- Entre el origen y el destino debe existir una conexión o canal por la que circulen los datos.
- La apertura de un archivo supone establecer la conexión del programa con el dispositivo que contiene el archivo.
- Hay tres flujos o canales abiertos automáticamente:
 - `extern FILE *stdin;`
 - `extern FILE *stdout;`
 - `extern FILE *stderr;`

5.2 Flujos (strams)

- Estas tres variables se inicializan al comenzar la ejecución del programa.
- Las tres admiten secuencias de caracteres en modo texto
- Tienen el siguiente cometido:
 - `stdin` asocia la entrada estándar (teclado) con el programa
 - `stdout` asocia la salida estándar (pantalla) con el programa
 - `stderr` asocia la salida de mensajes de error (pantalla) con el programa

5.2 Flujos (streams)

- El acceso a los archivos se hace con un buffer intermedio.
- Se puede pensar en el buffer como un array donde se van almacenando los datos dirigidos al archivo, o leídos desde el archivo
- El buffer se vuelca cuando de una forma u otra se da la orden de vaciarlo.
 - Por ejemplo, cuando se llama a una función para leer de un archivo una cadena, la función lee tantos caracteres como quepan en el buffer.

5.3 Apertura y cierre de un archivo

- Para comenzar a procesar un archivo en C la primera operación que hay que realizar es abrir el archivo.
- La apertura del archivo supone conectar el archivo externo con el programa, e indicar cómo va a ser tratado el archivo:
 - Binario
 - Texto
- El programa accede a los archivos a través de un puntero a la estructura FILE, la función de apertura devuelve dicho puntero.
- Esta estructura incluye entre otras cosas información sobre el nombre del archivo, la dirección de la zona de memoria donde se almacena el fichero, tamaño del buffer.

5.3 Apertura y cierre de un archivo

- La función para abrir un archivo es `fopen()`. Su formato es:

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

`nombre` es una cadena que contiene el identificador externo del archivo

`modo` es una cadena que contiene el modo en el que se va a tratar el fichero.

- La función `fopen()` devuelve un puntero a archivo. Un programa nunca debe alterar el valor de ese puntero.
- Si se produce un error cuando se está intentando abrir un archivo, `fopen()` devuelve un puntero nulo.

5.3 Apertura y cierre de un archivo

Modo	Significado
"r"	Abre un archivo de texto para lectura
"w"	Abre para crear un nuevo archivo de texto (si ya existe se pierden sus datos)
"a"	Abre un archivo de texto para añadir al final
"r+"	Abre archivo un archivo de texto ya existente para modificar (leer/escribir)
"w+"	Crea un archivo de texto para escribir/leer (si ya existe se pierden los datos)
"a+"	Abre el archivo de texto para modificar (escribir/leer) al final. Si no existe es como w+
"rb"	Abre un archivo binario para lectura
"wb"	Abre para crear un nuevo archivo binario (si ya existe se pierden sus datos)
"ab"	Abre un archivo binario para añadir al final
"r+b"	Abre archivo un archivo binario ya existente para modificar (leer/escribir)
"w+b"	Crea un archivo binario para escribir/leer (si ya existe se pierden los datos)
"a+b"	Abre el archivo binario para modificar (escribir/leer) al final. Si no existe es como w+

5.3 Apertura y cierre de un archivo

- Si se usa `fopen()` para abrir un archivo **para escritura (modo "w")**, entonces cualquier archivo existente con el mismo nombre se borrará y se crea uno nuevo.
 - OJO: Dependiendo del sistema de ficheros, importarán mayúsculas/minúsculas en el nombre:
 - Windows/MacOs: NTFS/Fat32/exFat...
 - No importan las mayúsculas/minúsculas
 - Linux: ext2/3/4
 - Sí importan las mayúsculas/minúsculas
- Si no existe un archivo con el mismo nombre, entonces se creará.
- Si se quiere añadir al final del archivo entonces debe usar el modo "a".
 - Si se usa "a" y no existe el archivo, se devolverá un error.
- La apertura de un archivo para las operaciones de lectura ("**r**") requiere que exista el archivo.
 - Si no existe, `fopen()` devolverá un error.
- Finalmente, si se abre un archivo para las operaciones de leer / escribir, la computadora no lo borrará si existe; sin embargo, si no existe, la computadora lo creará

Ejemplo 1

- Testear si un archivo existe:

```
#include <stdio.h>
#include <stdlib.h>

void main(){
    FILE *fd; //file descriptor
    fd = fopen("prueba.txt","r"); //intentamos abrirlo en modo lectura
    if (fd==NULL) //Comprobar si ha habido errores
        printf("Error, el archivo no existe\n");
    else
        printf("Error, el archivo existe\n");
    fclose (fd);
}
```

Revisar todas las posibles combinaciones con todos los modos de tratamiento de ficheros con la existencia o no del fichero objeto del tratamiento

5.3 Apertura y cierre de un archivo

- Al terminar la ejecución del programa podrá ocurrir que haya datos en el buffer de entrada/salida, y **si no se volcasen en el archivo quedaría éste sin las últimas actualizaciones.**
- Siempre que se termina de procesar un archivo y siempre que se termine la ejecución del programa **los archivos abiertos hay que cerrarlos** para:
 - Se vuelque al destino toda la información que todavía se encuentre en el buffer.
 - Realizar el cierre formal del archivo a nivel de sistema operativo
- Un error en el cierre puede generar problemas
 - Incluyendo la pérdida de datos
 - Destrucción de archivos
 - Posibles errores intermitentes en el programa

5.3 Apertura y cierre de un archivo

- La función `close ()` cierra el archivo asociado al `puntero_file`. Su formato es:

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

- Devuelve 0 si el cierre ha tenido éxito
- Devuelve EOP si ha habido un error al cerrar. Generalmente, esta función solo falla cuando un disco se ha retirado antes de tiempo o cuando no queda espacio libre en el mismo

5.4 Funciones de lectura y escritura

```
FILE *fopen (char *nombre_archivo, char *modo);  
int *fclose (FILE *puntero_file);  
int putc(int char, FILE *stream)  
int fputc(int char, FILE *stream)  
int getc(FILE *stream)  
int fgetc(FILE *stream)  
int fputs(const char *str, FILE *stream)  
char *fgets(char *str, int n, FILE *stream)  
char *fprintf(FILE *stream, const char *formato)  
int fscanf(FILE *stream, const char *formato)  
int feof(FILE *stream)  
void rewind(FILE *stream)
```

5.4 Funciones de lectura y escritura

- Las funciones `putc ()` y `fputc ()` son idénticas
 - Escriben un carácter en el archivo asociado con el puntero a `FILE`.
 - Devuelven el carácter escrito, o bien `EOF` si no puede ser escrito
 - Sus prototipos son:

```
int putc(int char, FILE *stream)
```

```
int fputc(int char, FILE *stream)
```

Ejemplo 2 `putc`

- Escribe en un fichero todas las Letras mayúsculas y minúsculas separadas por el carácter punto y coma (;)

5.4 Funciones de lectura y escritura

- Las funciones `getc ()` y `fgetc ()` son idénticas
 - Leen un carácter (el siguiente carácter) en el archivo asociado con el puntero a `FILE`.
 - Devuelven el carácter leído, o bien `EOF` si es fin de archivo (o si ha habido un error)
 - Sus prototipos son:

```
int getc(FILE *stream)
```

```
int fgetc(FILE *stream)
```



```
int putc(int char, FILE *stream)
```

Ejemplo 3 `getc`

- Lee un fichero del ejemplo anterior que contiene las letras en mayúsculas y minúsculas separadas por ‘;’ y presentarlo por pantalla

Ejercicio 1

- Basándose en el ejercicio anterior crear una tabla represente por columnas las letras tanto en mayúsculas como en minúsculas y su representación en ascii.

5.4 Funciones de lectura y escritura

- La función `fputs()`:
 - Escribe una cadena en el archivo asociado con el puntero a `FILE`, pero no escribe el carácter nulo (`'\0'`).
 - Devuelve `EOF` si no ha podido escribir la cadena y un valor no negativo si la escritura es correcta.
 - Su prototipo es:

```
int fputs(const char *str, FILE *stream)
```

Ejemplo 4. `fputs`

- Escribe el texto “hola mundo” en un fichero.

5.4 Funciones de lectura y escritura

- La función `fgets()`:
 - Lee una cadena de caracteres de un archivo asociado con el puntero a `FILE`.
 - Termina la lectura cuando lee el carácter fin de línea (`\n`) o cuando ha leído `n-1` caracteres, siendo `n` el segundo parámetro que se le pasa a la función
 - Devuelve un puntero a la cadena devuelta, o `NULL` si ha habido un error.
 - Su prototipo es:

```
char *fgets(char *str, int n, FILE *stream)
```

- `str` – Este es el puntero a un array de caracteres donde la cadena leída se almacena.
- `n` – Este es el máximo número de caracteres para ser leídos (incluyendo el carácter nulo final)
- `stream`. Este es el puntero a la estructura `FILE` que identifica el fichero de donde los caracteres van a ser leídos.

Ejemplo 5 `fgets`

- Recupera una cadena de caracteres escrita en un fichero e imprímela por pantalla.

5.4 Funciones de lectura y escritura

- Las funciones `printf ()` y `scanf ()` permiten escribir o leer variables de cualquier tipo de dato estándar, siendo los especificadores de formato (`%d`, `%f`, ...) los que indican la transformación que se debe realizar.
- La misma funcionalidad tienen `fprintf ()` y `fscanf ()`.
 - Sus prototipos son:

```
char *fprintf(FILE *stream, const char *formato)
```

```
int fscanf(FILE *stream, const char *formato)
```

- `stream`. Este es el puntero a la estructura `FILE` que identifica el fichero de donde los caracteres van a ser leídos.

Ejemplo 6 fprintf y fscanf

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main () {
    char str1[10], str2[10], str3[10];
    int year;
    FILE * fp;
```

```
    fp = fopen ("file.txt", "w+");
    fputs("Yo naci en es 2000", fp);
```

```
    rewind(fp);
    fscanf(fp, "%s %s %s %d", str1, str2, str3, &year);
```

```
    printf("Read String1 |%s|\n", str1 );
    printf("Read String2 |%s|\n", str2 );
    printf("Read String3 |%s|\n", str3 );
    printf("Read Integer |%d|\n", year );
```

```
    fclose(fp);
```

```
    return(0);
```

```
}
```

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main () {
    FILE * fp;
```

```
    fp = fopen ("file.txt", "w+");
    fprintf(fp, "%s %s %s %d", "We", "are", "in", 2012);
```

```
    fclose(fp);
```

```
    return(0);
```

```
}
```


5.4 Funciones de lectura y escritura

- La función `feof ()`

- Devuelve un valor distinto de 0 (true) cuando se lee el carácter de fin de archivo, en caso contrario devuelve 0 (false).

- Su prototipo es:

```
int feof(FILE *stream)
```

- `stream`. Este es el puntero a la estructura `FILE` que identifica el fichero de donde los caracteres van a ser leídos.

- La función `rewind ()`

- Sitúa el puntero del fichero al inicio del mismo.

- Su prototipo es:

```
void rewind(FILE *stream)
```

- `stream`. Este es el puntero a la estructura `FILE` que identifica el fichero de donde los caracteres van a ser leídos.

Ejemplo 7 feof y rwind

Ejercicio 2

- Realizar un programa que muestre las N primeras líneas de un fichero de texto.
- Se pasan como parámetros del main N y el nombre del fichero

```
$ mostrarLineas 4 fich.txt
```

- Mostraría las 4 primeras línea del fichero de texto fich.txt
- Es recomendable
 - Implementar una estructura para almacenar todo el texto con los siguientes miembros:

```
typedef struct lineas_t{  
    int Numlineas;  
    char **Lineas;  
}lineas_t;
```

- Implementar una función que lea las líneas del fichero. Su prototipo puede ser:

```
char *leelineaDinamicaFichero (FILE *fd)
```

Ejercicio 3

- Realizar un programa que muestre intercambie los dos primeros párrafos de un fichero. Se pasan como parámetros del main los números de párrafos a intercambiar

```
$ intercambiarLineas 4 2
```