Tema 1. Ficheros

FUNDAMENTOS DE PROGRAMACIÓN II

Profesor: Fernando Pereñíguez García

Escuela Politécnica

Contenidos

- La librería estándar de C.
- Entrada/salida.
  - Streams. Punteros a ficheros.
  - Streams estándares.
  - Redirección.
  - Ficheros de texto y ficheros binarios.
- Trabajando con ficheros.
  - Operaciones generales.
  - E/S: formateada, de carácter, de líneas, de bloques, de cadenas.
  - Posicionamiento en ficheros.

Contenidos

- **La librería estándar de C.**
- Entrada/salida.
  - Streams. Punteros a ficheros.
  - Streams estándares.
  - Redirección.
  - Ficheros de texto y ficheros binarios.
- Trabajando con ficheros.
  - Operaciones generales.
  - E/S: formateada, de carácter, de líneas, de bloques, de cadenas.
  - Posicionamiento en ficheros.

La librería estándar de C

- La librería estándar de C (C89) está dividida en 15 partes.
- Cada una de las partes está indetificada a travésde un fichero cabecera (header, .h).
- C99 añade 9 más, haciendo un total de 24 ficheros:

```
<assert.h>          <inttypes.h> (C99)      <signal.h>          <stdlib.h>
<complex.h> (C99)   <iso646.h> (C99)        <stdarg.h>          <string.h>
<ctype.h>           <limits.h>              <stdbool.h> (C99)   <tgmath.h> (C99)
<errno.h>           <locale.h>              <stddef.h>          <time.h>
<fenv.h> (C99)      <math.h>                <stdint.h> (C99)    <wchar.h> (C99)
<float.h>           <setjmp.h>              <stdio.h>           <wctype.h> (C99)
```

- Muchos compiladores actuales integran además un gran conjunto de librerías, las citadas en la lista anterior son las estándares.

Librerías no estándares

- Entre las librerías que no forman parte del estándar hay librerías que:
  - Proporcionan más control sobre el teclado y la pantalla.
  - Implementan interfaces gráficas basadas en ventanas.
  - Implementan primitivas gráficas.
- Para utilizar las librerías basta con:
  - Utilizar la cláusula **`#include`**
  - En algunos casos linkar con la librería.

Contenidos

- La librería estándar de C.
- **Entrada/salida.**
  - Streams. Punteros a ficheros.
  - Streams estándares.
  - Redirección.
  - Ficheros de texto y ficheros binarios.
- Trabajando con ficheros.
  - Operaciones generales.
  - E/S: formateada, de carácter, de líneas, de bloques, de cadenas.
  - Posicionamiento en ficheros.

Entrada/Salida

- La librería de E/S es la parte más grande e importante de la librería estándar de C.
- Hasta ahora se ha visto parte de stdio.h, concretamente funciones tales como:
  - printf, scanf, putchar, getchar, puts, gets...

... pero hay más funciones.

- Es importante destacar que la E/S a través de la librería estándar **utiliza buffers** para mejorar el rendimiento de las operaciones.

Flujos (I)

(Streams)

- En C, el término **stream** representa cualquier fuente de entrada de datos o cualquier destino de salida de datos.
  - Muchos de los programas pequeños, como los ejemplos de la asignatura, obtienen sus datos desde un flujo de entrada, usualmente el teclado, y, escriben su salida en otro flujo, usualmente la pantalla.
  - Otros programas más grandes necesitan utilizar otros streams como por ejemplo:
    - Ficheros (en discos duros, DVD, CD, memorias USB, etc.).
    - Puertos de red.
    - Impresoras.
    - Etc.

Streams (II)

- La librería stdio.h proporciona varias funciones que son capaces de trabajan con varios tipos de streams, no únicamente los basados en ficheros.
- Sobre los streams en C caben destacar los siguientes conceptos:
  - **Punteros a ficheros (File Pointers).**
  - **Flujos estándar. Redirección.**
  - **Ficheros de texto y ficheros binarios.**

Contenidos

- La librería estándar de C.
- Entrada/salida.
  - **Streams. Punteros a ficheros.**
  - Streams estándares.
  - Redirección.
  - Ficheros de texto y ficheros binarios.
- Trabajando con ficheros.
  - Operaciones generales.
  - E/S: formateada, de carácter, de líneas, de bloques, de cadenas.
  - Posicionamiento en ficheros.

Punteros a ficheros (III):

(File Pointers)

- El acceso a un flujo dentro de la librería estándar de C se realiza a través de un puntero a una estructura del tipo FILE.
  - El tipo FILE se declara en stdio.h.
- Si un programa necesita acceder a dos streams basta que haga las siguientes declaraciones:

FILE *fp1, *fp2;

- Un programa puede declarar cualquier número de streams que desee, el sistema operativo siempre impone un límite de streams que un programa puede tener abiertos en un momento dado.

Contenidos

- La librería estándar de C.
- Entrada/salida.
  - Streams. Punteros a ficheros.
  - **Streams estándares.**
  - Redirección.
  - Ficheros de texto y ficheros binarios.
- Trabajando con ficheros.
  - Operaciones generales.
  - E/S: formateada, de carácter, de líneas, de bloques, de cadenas.
  - Posicionamiento en ficheros.

Streams estándares (IV)

- <stdio.h> proporciona tres streams stándares:

Nombre	Stream	Significado por defecto
stdin	Standard input	Teclado
stdout	Standard output	Pantalla
stderr	Standard error output	Pantalla

- Cualquier proceso que se ejecuta siempre tendrá acceso a esos tres streams, no se declaran, no se abren y no se cierran.

Contenidos

- La librería estándar de C.
- Entrada/salida.
  - Streams. Punteros a ficheros.
  - Streams estándares.
  - **Redirección.**
  - Ficheros de texto y ficheros binarios.
- Trabajando con ficheros.
  - Operaciones generales.
  - E/S: formateada, de carácter, de líneas, de bloques, de cadenas.
  - Posicionamiento en ficheros.

Redirección (V)

- Hasta ahora, todas las funciones utilizadas: printf, scanf, putchar, getchar, puts y gets, obtenían la entrada de stdin y mostraban la salida en stdout.

- Por defecto, stdin representa el teclado y, stdout y stderr la pantalla. Este comportamiento se puede cambiar a través de los **mecanismos de redireccionamiento**:

programa > out.dat

- El programa redireccionará toda la salida al fichero out.dat, nada se mostrará por pantalla.

programa < in.dat

- El programa tomará la entrada del fichero in.dat, no se leerá del teclado.

programa < in.dat > out.dat

- El programa tomará la entrada del fichero in.dat volcando la salida al fichero out.dat.

Contenidos

- La librería estándar de C.
- Entrada/salida.
  - Streams. Punteros a ficheros.
  - Streams estándares.
  - Redirección.
  - **Ficheros de texto y ficheros binarios.**
- Trabajando con ficheros.
  - Operaciones generales.
  - E/S: formateada, de carácter, de líneas, de bloques, de cadenas.
  - Posicionamiento en ficheros.

Ficheros de texto y ficheros binarios (VI)

- <stdio.h> soporta dos tipos de ficheros:
  - **Ficheros de texto.**
    - Los byte de un fichero de texto representan caracteres, de tal manera que se pueden examinar y editar directamente.
    - Los caracteres deben seguir una codificación estándar (UTF-8, ISO-8859, etc.).
    - Muy usado en programación. Los programas en C se escriben en ficheros de texto.
  - **Ficheros binarios.**
    - No necesariamente representan texto. Un fichero binario es un conjunto de bytes con significado para el sistema operativo (como un fichero ejecutable o una DLL) o para cualquier aplicación (un fichero de Word tiene sentido para Microsoft Word).

Ficheros de texto y ficheros binarios (VII)

- Los ficheros de texto tienen **dos características** que no poseen los ficheros binarios:

1.- Están divididos en líneas.

- Cada una de las líneas que conforman el fichero de texto terminan con uno o dos caracteres especiales.
- El carácter de terminación depende del sistema operativo:
  - Windows: formado por dos caracteres, un retorno de carro seguido de un salto de línea (0x0a + 0x0d).
  - UNIX y MacOS: formado por un salto de línea (0x0a).

2.- Pueden contener un carácter especial fin de fichero.

- Algunos sistemas operativos pueden utilizar un carácter especial para marcar el final del fichero.
- Muchos sistemas operativos modernos, incluyendo UNIX no lo utilizan.

Contenidos

- La librería estándar de C.
- Entrada/salida.
  - Streams. Punteros a ficheros.
  - Streams estándares.
  - Redirección.
  - Ficheros de texto y ficheros binarios.
- **Trabajando con ficheros.**
  - Operaciones generales.
  - E/S: formateada, de carácter, de líneas, de bloques, de cadenas.
  - Posicionamiento en ficheros.

Trabajando con ficheros (I)

- La redirección, a pesar de ser un mecanismo simple, no permite un control absoluto sobre los ficheros.
- Trabajar con un fichero siempre implica tres pasos:

  1.- Abrir el fichero.

  2.- Leer y/o escribir en él.

  3.- Cerrar el fichero.

Trabajando con ficheros (II)

- Para trabajar con ficheros, la librería estándar nos proporciona un gran conjunto de funciones.
- Estas funciones las podemos clasificar en:
  - Operaciones generales sobre ficheros.
  - Entrada/salida formateada.
  - Entrada/salida de carácter.
  - Entrada/salida de líneas.
  - Entrada/salida de bloques.
  - Posicionamiento en ficheros.
  - Entrada/salida de cadenas.

Contenidos

- La librería estándar de C.
- Entrada/salida.
  - Streams. Punteros a ficheros.
  - Streams estándares.
  - Redirección.
  - Ficheros de texto y ficheros binarios.
- Trabajando con ficheros.
  - **Operaciones generales.**
  - E/S: formateada, de carácter, de líneas, de bloques, de cadenas.
  - Posicionamiento en ficheros.

Operaciones generales: fopen (I)

FILE *fopen(const char *path, const char *nombre);

- Antes de poder utilizar un stream, es obligatorio abrirlo con la función fopen.
  - La función necesita el nombre del fichero y un modo.
  - La función devuelve un puntero a FILE. Sucesivas operaciones sobre el fichero requerirán este valor.
- El argumento path representa el nombre del fichero, puede ser una ruta absoluta o relativa.
- Podemos abrir un fichero en Windows con la llamada:

```
fp = fopen(C:\\proyecto\\fichero.dat., .r.);
```

- En UNIX, la misma llamada sería:

```
fp = fopen(./proyecto/fichero.dat., .r.);
```

Modos de fopen (II)

- La forma de abrir un fichero con fopen depende de:
  - Las operaciones que se van a realizar sobre él.
  - Si el fichero es un fichero de texto o binario.
- Los **modos** para un fichero de texto son:

"r" Abre un fichero para lectura.

"w" Abre un fichero para escritura, si no existe lo crea.

"a" Abre un fichero para añadir datos, si no existe lo crea.

"r+" Abre un fichero para lectura y escritura, desde el principio del fichero.

"w+"Abre un fichero para lectura y escritura, si el fichero existe se trunca todo su contenido.

"a+" Abre un fichero para lectura y escritura (añadiendo si el fichero existe).

Modos de fopen (III)

- El modo puede añadir una 'b' para indicar que se va a abrir un fichero binario.
  - Se mantiene por razones de compatibilidad con C89, en Linux se ignora.
  - Se puede añadir por razones de portabilidad con otros sistemas.
- Los modos quedarían entonces así:

"rb" Abre un fichero para lectura.

"wb" Abre un fichero para escritura, si no existe lo crea.

"ab" Abre un fichero para añadir datos, si no existe lo crea.

"rb+" Abre un fichero para lectura y escritura, desde el principio del fichero.

"wb+" Abre un fichero para lectura y escritura, si el fichero existe se trunca todo su contenido.

"ab+" Abre un fichero para lectura y escritura (añadiendo si el fichero existe).

Operaciones generales: fclose (IV)

int fclose(FILE *fp);

- Permite cerrar un fichero.
- La función devuelve:
  - Cero si el fichero se cierra correctamente.
  - Un código de error.

Operaciones generales: remove (V)

int remove(const char *pathname);

- Borra un fichero.
- La función devuelve:
  - Cero si el fichero se elimina correctamente.
  - Un código de error.
- Ejemplo:

```
remove("/tmp/foto.jpg");
```

Operaciones generales: rename (y VI)

int rename(const char *oldpath, const char newpath);

- Renombra un fichero.
- La función devuelve:
  - Cero si el fichero se elimina correctamente.
  - Un código de error.
- Si el fichero a renombrar está abierto, hay que cerrarlo antes de utilizar la función rename.

Contenidos

- La librería estándar de C.
- Entrada/salida.
  - Streams. Punteros a ficheros.
  - Streams estándares.
  - Redirección.
  - Ficheros de texto y ficheros binarios.
- Trabajando con ficheros.
  - Operaciones generales.
  - **E/S: formateada, de carácter, de líneas, de bloques, de cadenas.**
  - Posicionamiento en ficheros.

Entrada/salida formateada (I)

int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);

- La función fprintf funciona exactamente igual que printf pero permite realizar la salida sobre el stream definido a través del argumento FILE.
  - Ejemplo:

```
printf("Total: %d\n", total);
```

  - ... muestra el mansaje en pantalla, mientras que:

```
fprintf(fp, "Total: %d\n", total);
```

  - ...escribe el mensaje en el stream representado por fp.

Entrada/salida formateada (II)

int scanf(const char *format, ...);

int fscanf(FILE *stream, const char *format, ...);

- La función fscanf funciona exactamente igual que scanf pero permite realizar la entrada sobre el stream definido a través del argumento FILE.
- Ambas funciones devuelven el número de elemento leídos.

Entrada/salida formateada (y III)

void clearerr(FILE *stream);
int feof(FILE *stream);
int ferror(FILE *stream);

- Si pedimos leer de un stream n items, pero leemos menos, es que algo ha ido mal. Tres son las posibilidades:
  - End-of-file. La función se encontró con el final del fichero antes de completar la petición de lectura.
  - Error de lectura. La función no pudo leer caracteres del stream.
  - Error de coincidencia. El ítem leído estaba en un formato erróneo.
- **`feof`** comprueba si se ha alcanzado el final del fichero.
- **`ferror`** comprueba si se ha producido un error.
- **`clearerr`** inicializa los indicadores de error y fin de fichero.

Entrada/salida de carácter (I)

int fputc(int c, FILE *stream);
int putc(int c, FILE *stream);
int putchar(int c);

- putchar muestra un carácter en la salida estándar.
- fputc y putc son funciones más generales que escriben un carácter en el stream que se indique.
- La función devuelve:
  - Si ocurre algo, devuelven EOF y fijan un indicador de error.
  - Si todo va bien, devuelven el carácter mostrado/escrito.

Entrada/salida de carácter (y II)

int fgetc(FILE *stream);
int getc(FILE *stream);
int getchar(void);

- getchar lee un carácter de la entrada estándar.
- fgetc y getc son funciones más generales que leen un carácter del stream que se indique.
- La función devuelve:
  - Si ocurre algo, devuelven EOF y fijan un indicador de error.
    - Por esta razón las funciones devuelven un carácter unsigned char que se convierte a través de un casting en int.
  - Si todo va bien, devuelven el carácter mostrado/escrito.

Entrada/salida de líneas (I)

int fputs(const char *s, FILE *stream);
int puts(const char *s);

- Estas funciones son más adecuadas para streams de texto, pero también se pueden usar con streams binarios.
- **`puts`** muestra una línea en la salida estándar.
  - puts muestra además el retorno de carro.
- **`fputs`** es una función más general que escribe una línea en el stream que se indique.
  - fputs no escribe el retorno de carro a no ser que la cadena lo incluya.
- La función devuelve:
  - Si ocurre algo, devuelven EOF, sino un número positivo.

Entrada/salida de líneas (y II)

char *fgets(char *s, int size, FILE *stream);
char *gets(char *s);

- **`gets`** no debe utilizarse, ya que no comprueba el tamaño del array que se pasa como argumento.
- **`fgets`** es una función más general que lee una línea del stream que se indique.
  - **`fgets`** lee **size - 1** caracteres a no ser que se encuentre un retorno de carro antes. Almacena los caracteres, si lee el retorno de carro, también lo almacena.
  - Al final de la cadena, almacena un carácter '\0'.
- La función devuelve:
  - Si ocurre algo un puntero a NULL.
  - Si todo va bien, un puntero a la cadena leída.

Entrada/salida de bloques

size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);

size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);

- Las funciones **`fread`** y **`fwrite`**, permiten leer/ escribir de una sola vez un conjunto de datos grandes.
  - Se utilizan sobretodo sobre flujos binarios, pero también se pueden utilizar sobre flujos de texto.
- Ambas funciones devuelven el número de elementos leídos/escritos.

Contenidos

- La librería estándar de C.
- Entrada/salida.
  - Streams. Punteros a ficheros.
  - Streams estándares.
  - Redirección.
  - Ficheros de texto y ficheros binarios.
- Trabajando con ficheros.
  - Operaciones generales.
  - E/S: formateada, de carácter, de líneas, de bloques, de cadenas.
  - **Posicionamiento en ficheros.**

Posicionamiento en ficheros

int fgetpos(FILE *stream, fpos_t *pos);

int fsetpos(FILE *stream, fpos_t *pos);

int fseek(FILE *stream, long offset, int whence);

long int ftell(FILE *stream);

void rewind(FILE *stream);

- **Cada stream tiene asociado una posición de fichero**.
  - Cuando se abre un fichero, el apuntador se coloca al principio. Si un fichero se abre para añadir, el puntero se posiciona al final.
- Estas funciones permiten manejar el apuntador a la posición actual del fichero.

Posicionamiento en ficheros

RESUMEN FUNCIONES POSICIONAMIENTO
`int fgetpos(FILE *stream, fpos_t *pos);`
Devuelve posición actual del fichero (se almacena en “pos”)
`int fsetpos(FILE *stream, fpos_t *pos);`
Establece posición actual al valor indicado en “pos”
`void rewind(FILE *stream);`
Coloca el indicador de posición de fichero al comienzo del mismo.
`int fseek(FILE *stream, long offset, int whence);`
Modifica posición del fichero al valor “whence + offset”. “whence” puede ser: SEEK_SET, SEEK_CUR, SEEK_END.
`long int ftell(FILE *stream);`
Devuelve posición actual del fichero, respecto al comienzo, en caracteres.

Entrada/salida de cadenas

int sprintf(char *str, const char *format, ...);

int snprintf(char *str, size_t size, const char *format, ...);

int sscanf(const char *str, const char *format, ...);

- Estas funciones no pueden manejar streams, pero tienen la habilidad de mostrar/leer desde cadenas como si éstas fuesen streams.

Ejemplo: función para leer una cadena de la entrada estándar

```
int leer_string(char *cadena, int tam)
{
    char buf[LINE_MAX];
    char *p;
    fgets(buf, LINE_MAX, stdin);
    if ((p = strchr(buf, '\n')) != NULL)
        *p = '\0';
    strncpy(cadena, buf, tam - 1);
    cadena[tam - 1 ] = '\0';
}
```

Ejemplo: función para leer un entero, int, de la entrada estándar

```
int leer_int(int *d)
{
    char buf[LINE_MAX];
    fgets(buf, sizeof(buf), stdin);
    return sscanf(buf, "%d", d);
}
```

Ejemplo:copiar un fichero con getc/putc, pasando el origen y el destino de la copia como argumentos

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char** argv)
{
    FILE *fuente, *destino;
    int ch;
    if (argc != 3) {
        fprintf (stderr, "Uso: %s ficheroFuente ficheroDestino\n", argv [0]);
        exit (EXIT_FAILURE);
    }
    if ((fuente = fopen(argv[1], "rb")) == NULL) {
        fprintf(stderr, "No se puede abrir el fichero fuente %s\n", argv[1]);
        exit (EXIT_FAILURE);
    }
    if ((destino = fopen(argv[2], "wb")) == NULL) {
        fprintf(stderr, "No se puede abrir el fichero destino %s\n", argv[2]);
        fclose(fuente);
        exit (EXIT_FAILURE);
    }
    while ((ch = getc(fuente)) != EOF)
        putc(ch, destino);
    fclose(fuente);
    fclose(destino);
    return (EXIT_SUCCESS);
}
```

Ejemplo:copiar un fichero con fread/ fwrite, pasando el origen y el destino de la copia como argumentos

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char** argv)
{
    FILE *fuente, *destino;
    int ch;
    if (argc != 3) {
        fprintf (stderr, "Uso: %s ficheroFuente ficheroDestino\n", argv [0]);
        exit (EXIT_FAILURE);
    }
    if ((fuente = fopen(argv[1], "rb")) == NULL) {
        fprintf(stderr, "No se puede abrir el fichero fuente %s\n", argv[1]);
        exit (EXIT_FAILURE);
    }
    if ((destino = fopen(argv[2], "wb")) == NULL) {
        fprintf(stderr, "No se puede abrir el fichero destino %s\n", argv[2]);
        fclose(fuente);
        exit (EXIT_FAILURE);
    }
    while (fread(&ch, sizeof (c), 1, origen) > 0)
        fwrite(&ch, sizeof (c), 1, destino);
    fclose(fuente);
    fclose(destino);
    return (EXIT_SUCCESS);
}
```

Ejemplo: generar un fichero de texto con un contenido

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char** argv)
{
    FILE *destino;
    int ch;
    if (argc != 2) {
        fprintf (stderr, "Uso: %s ficheroDestino\n", argv [0]);
        exit (EXIT_FAILURE);
    }
    if ((destino = fopen(argv[1], "w")) == NULL) {
        fprintf(stderr, "No se puede abrir el fichero destino %s\n", argv[1]);
        exit (EXIT_FAILURE);
    }
    fputs("Estamos creando un fichero de texto...\n", destino);
    fputs("con este contenido.\n", destino);
    fprintf(destino, “%d + %d = %d \n”, 3, 4, 7);
    fputs("Aquí terminamos.\n", destino);
    fclose(destino);
    return (EXIT_SUCCESS);
}
```

Ejemplo: almacenar el contenido de una estructura en un fichero

```
struct coche {
    char marca [15];
    char modelo [20];
    int cilindrada;
    int potencia;
    int velocidad_maxima;
};
struct garaje_type {
int tam;
struct coche coches [TAM_GARAJE];
};
...
struct garaje_type *g;
FILE *fp;
...
fwrite(g, sizeof(struct garaje_type), 1, fp);
...
fread(g, sizeof(struct garaje_type), 1, fp);
```

fread y fwrite

```
include <stdio.h>
int main(){
    FILE *fichero;
    char nombre[11] = "datos5.txt";
    unsigned int dinero[10] = { 23, 12, 45, 345, 512, 345, 654, 287, 567, 124 };
    unsigned int leer[10], i;
    fichero = fopen( nombre, "w+" );
    printf( "Fichero: %s -> ", nombre );
    if( fichero )  printf( "creado (ABIERTO)\n" );
    else { printf( "Error (NO ABIERTO)\n" ); return 1;  }

    printf( "Escribiendo cantidades:\n\n" );
    for( i=0; i<10; i++ )    printf( "%d\t", dinero[i] );
    fwrite( dinero, sizeof(unsigned int), 10, fichero );

    printf( "\nLeyendo los datos del fichero \"%s\":\n", nombre );   rewind( fichero );
    fread( leer, sizeof(unsigned int), 10, fichero );
    for( i=0; i<10; i++ ) printf( "%d\t", leer[i] );

    if( !fclose(fichero) ) printf( "\nFichero cerrado\n" );
    else { printf( "\nError: fichero NO CERRADO\n" ); return 1;   }
    return 0;
}
```

Bibliografía

- King, K.N. **C Programming. A modern approach.** 2ªed. Ed. W.W. Norton & Company. Inc. 2008. Chapter 22.
- Khamtane Ashok**. Programming in C**. Ed. Pearson. 2012.
- Ferraris Llanos, R. D. **Fundamentos de la Informática y Programación en C**. Ed. Paraninfo. 2010.