uc3m | Universidad Carlos III de Madrid

# Informática Industrial I

Grado en Ingeniería en Electrónica Industrial y Automática

Álvaro Castro González

Nicolas Burus

Mohamed Abderrahim

José Carlos Castillo Montoya

## Práctica 7

## Ficheros

A lo largo de los últimos años, han colaborado en impartir las prácticas presenciales, y por tanto proporcionar comentarios para mejorar algunos aspectos de este documento los profesores siguientes:

Ioannis Douratsos
Fares Abu-Dakka
Juan Carlos González Victores
Raúl Perula
Avinash Ranganath

# Práctica 7 – Ficheros.

## 1. Apertura y cierre de ficheros

En esta práctica se verá cómo trabajar con ficheros del sistema. Hay que tener en cuenta que los ficheros de I/O (Input/Output) son simplemente un caso especial del sistema general de I/O del sistema y por lo tanto lo que se va a ver que se puede aplicar también a cualquier otro flujo de datos conectado a otros tipos de dispositivos.

Para trabajar con ficheros en C++ es necesario incluir el fichero de cabecera: `fstream`.

Lo primero que hay que hacer para abrir un fichero es crear un flujo que dependerá de la "dirección" de los datos. Hay tres tipos de flujos:

- De entrada: `ifstream`
- De salida: `ofstream`
- De entrada/salida: `fstream`

Para abrir un fichero se llamará a la función `open()` o directamente mediante el constructor correspondiente. Esta función recibe como parámetros la ruta del fichero y el modo de apertura. El modo se determina con uno o más de estos valores:

- `ios::app` → añadir datos al final del fichero.
- `ios::ate` → posicionarse al final del fichero.
- `ios::in` → determina un flujo de entrada.
- `ios::out` → determina un flujo de salida.
- `ios::binary` → modo binario (por defecto está el modo texto).
- `ios::trunc` → cambia el tamaño del fichero a 0 eliminando los datos.

Esta función tiene asignados varios valores por defecto, con lo que no estrictamente necesario ponerlos.

A continuación se puede ver un ejemplo de uso:

```
#include <iostream>
#include <fstream>

Using namespace std;

int main(int argc, char *argv[]) {
      ifstream input("input.txt"); //reading file in text mode
      if(!input){
            cerr << "Error opening input file!" << endl;
      }

      ofstream output;
      //writing file in binary mode
      output.open("output.bin", ios::out | ios::binary);
      if(output.is_open()) { //check if file is opened
            cerr << "Output file ready" << endl;
      }

      // ... do things with files ...
```

```
	input.close();
	output.close();

	return 0;
}
```

## 2. Lectura y escritura de ficheros de texto

Para leer y escribir ficheros de texto se pueden usar directamente los operadores >> y << respectivamente, igual que si se hiciera por consola. Por ejemplo se podría hacer algo como:

```
output << "escribiendo en el flujo de salida output" << endl;
```

**Ejercicio**

*Escribir un programa que pida al usuario el nombre y apellidos de varias personas y los escriba en un fichero usando una línea para los datos de cada persona. A continuación hacer otro programa que los lea y los muestre por pantalla.*

¿Cómo detectar el final del fichero? Se puede usar la función `eof()` (end of file) del flujo correspondiente o tener en cuenta que cuando se llega al final de fichero el flujo asociado devuelve `false`).

Otra interesante función que permite la lectura de datos es `getline()` cuyos prototipos son:

```
istream& getline(char *buf, streamsize num);
istream& getline(char *buf, streamsize num, char delim);
```

Estas funciones leen caracteres del flujo y los almacenan en el array apuntado por `buf` hasta que se haya leído `num-1` caracteres o hasta que se encuentra un final de línea (el carácter `delim` en la segunda forma) o hasta que se llegue al final de fichero. El carácter delimitador se extraerá y no se añadirá al array `buf`.

La función `get()` tiene otras dos formas que funcionan de manera muy parecida a `getline`. Estos son los prototipos:

```
istream& get(char *buf, streamsize num);
istream& get(char *buf, streamsize num, char delim);
intget();
```

La única diferencia entre las dos primeras formas y `getline` es que las formas de `get` no eliminan el delimitador y permanece en el flujo hasta la siguiente operación de lectura. En cuanto a `get()` devuelve el siguiente carácter y en caso de llegar al final de fichero devolverá `EOF`.

**Ejercicio**

*Haz un programa que se encargue de leer el fichero que anteriormente has creado de nombres y los muestre por pantalla línea por línea.*

## 3. Ficheros binarios

Los ficheros de texto no son siempre los más eficaces. Además puede ser que se necesite escribir datos que no tienen formato, es decir, en crudo (raw). Por estos motivos es necesario poder trabajar con ficheros binarios.

Para trabajar con ficheros binarios es necesario abrir el fichero en el siguiente modo (`ios::binary`).

Para leer y escribir un fichero binario se puede hacer byte a byte, usando las funciones `get()` y `put()`. O se puede hacer por bloques, usando las funciones `read()` y `write()`.Que se encargan respectivamente de leer/escribir `num` caracteres desde la posición apuntada por `buf`.

```
istream& read(char *buf, streamsizenum);
ostream& write(char *buf, streamsizenum);
```

**Ejercicio**

*Crear un programa que escriba carácter a carácter todos los caracteres del 0 al 255 en un fichero binario. Leer dicho fichero y mostrar por pantalla los números pares.*

## 4. Acceso aleatorio

En C++ existen dos punteros asociados a los ficheros: uno es el puntero `get`, que apunta a la posición donde se va a realizar la siguiente operación de entrada (lectura), y el otro es el puntero `put`, que especifica la siguiente posición donde se realizará la próxima operación de salida (escritura).

Para modificar estos punteros existen las funciones `seekg` y `seekp`:

```
istream & seekg(off_type offset, seekdir origin);
ostream & seekp(off_type offset, seekdir origin);
```

Estas funciones desplazan sus respectivos punteros con un `offset` desde la posición indicada por `origin`, la cual puede ser uno de los siguientes valores:

- `ios::beg` → el comienzo del fichero.
- `ios::cur` → la posición actual.
- `ios::end` → el final del fichero.

Las operaciones de acceso aleatorio a un fichero generalmente sólo se utilizan en operaciones binarias y hay que ser especialmente cuidadoso puesto que puede provocar una falta de sincronización con los datos del fichero.

Para determinar la posición actual de los punteros se usan las funciones:

```
pos_typeseekg();
pos_typeseekp();
```

Cuyos resultados se pueden aplicar directamente a las funciones `seekg` y `seekp` de la siguiente manera:

```
istream & seekg(pos_type pos);
istream & seekp(pos_type pos);
```

**Ejercicio**

*Escribir un programa que guarde en formato binario los números enteros del 0 al 10 en un fichero. A continuación leerlos del fichero e imprimirlos por pantalla. Finalmente desplazar el puntero* `get` *del fichero un carácter desde el comienzo del fichero y volver a leerlos. ¿Qué resultado se obtiene? ¿Por qué?*

## Ejercicios

Continuar ampliando la funcionalidad del simulador de ordenador. En este caso se deberá crear dos nuevas clases `InputFile` y `OutputFile` que heredarán de `Input` y `Output` respectivamente.

`InputFile` se encargará de leer un fichero de texto y enviarlo al procesador correspondiente. A su vez crear otras dos clases que hereden de `InputFile` y que se comporten de forma diferente: una enviará los datos línea a línea y la otra palabra a palabra.

`OutputFile` escribirá cadenas en un fichero de texto añadiéndolas al final. El formato de cada línea será `[fecha+hora]:[cadena de texto]`.

Para obtener la fecha y la hora se podrá usar algo similar al siguiente código:

```
#include <time.h>
#include <string.h> // strtok: used to remove the line scape (‘\n’)

...

time_t rawtime;
struct tm *timeinfo;
time(&rawtime);
timeinfo = localtime(&rawtime);
cout << “TIME: “ << strtok(asctime(timeinfo), ”\n”) << endl;
```

La salida en el fichero deberá ser algo parecido a lo siguiente:

```
MonSep 06 18:39:05 2010:Processed by Uppercaseprocessor: EN
MonSep 06 18:39:05 2010:Processed by Uppercaseprocessor: UN
MonSep 06 18:39:05 2010:Processed by Uppercaseprocessor: LUGAR
MonSep 06 18:39:05 2010:Processed by Uppercaseprocessor: DE
MonSep 06 18:39:05 2010:Processed by Uppercaseprocessor: LA
MonSep 06 18:39:05 2010:Processed by Uppercaseprocessor: MANCHA,
MonSep 06 18:39:05 2010:Processed by Uppercaseprocessor: DE
```