



## Guion de prácticas

*Uso básico de ficheros  
Paso de parámetros a main*

*Abril de 2018*



## Metodología de la Programación

Curso 2017/2018



# Índice

<b>1. Descripción</b>	<b>5</b>
<b>2. Objetivos</b>	<b>5</b>
<b>3. Manejo de ficheros de texto</b>	<b>5</b>
3.1. Procedimiento para leer datos desde un fichero . . . . .	5
3.2. Procedimiento para escribir datos en un fichero . . . . .	6
3.3. Gestión básica de errores con ficheros . . . . .	8
<b>4. Paso de parámetros a Main desde la línea de órdenes</b>	<b>9</b>
<b>5. Práctica a entregar: gestionar idiomas</b>	<b>10</b>
5.1. Corrección de la práctica . . . . .	11
<b>6. Apéndice 1. Código de ejemplo</b>	<b>13</b>
6.1. idioma.h . . . . .	13



## 1. Descripción

Por ahora los únicos datos con los que pueden trabajar nuestros programas son los que introducimos desde el teclado o los que mostramos por la pantalla, pero en programas reales esto no es suficiente, sino que se hace necesario almacenar de forma permanente tanto los datos de entrada como los de salida de un mismo programa, para lo que se recurre al uso de ficheros en memoria masiva. En esta práctica introduciremos muy brevemente el protocolo de manejo de ficheros de texto, dejando los ficheros binarios y los detalles más avanzados del uso de ficheros para su introducción en temas posteriores.

## 2. Objetivos

- Conocer la estructura de los ficheros de texto.
- Leer datos de un fichero de texto.
- Escribir datos en un fichero de texto.
- Practicar el paso de parámetros a **main()** desde la línea de órdenes

## 3. Manejo de ficheros de texto

Los ficheros de texto contienen datos que han sido codificados como texto usando un esquema como ISO8859-1<sup>1</sup> o UTF<sup>2</sup> y, a diferencia de los ficheros binarios, aparecen exactamente como en la pantalla del ordenador, es decir, como una secuencia de caracteres. Por tanto un fichero de texto puede abrirse y editarse con programas editores como `notepad` en Windows o `gedit` en Ubuntu Linux.

### 3.1. Procedimiento para leer datos desde un fichero


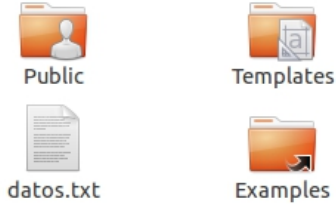
Los datos de un fichero se pueden procesar de forma muy similar a como se leen datos desde el teclado o como se escriben datos en pantalla con los operadores de extracción `>>` y de inserción `<<` en un flujo de datos. En el Cuadro 1 se pueden ver dos programas que leen exactamente los mismos valores para cada variable con la diferencia de que el primero los lee desde el teclado y el segundo los lee desde un fichero llamado "datos.txt".

Se puede ver que el procedimiento para leer los datos desde un fichero es muy sencillo y consiste en los siguientes pasos.

1. Incluir el fichero de cabeceras `fstream` para poder manejar ficheros.

<sup>1</sup>Codificación ISO8859-1 [https://en.wikipedia.org/wiki/ISO/IEC\\_8859-1](https://en.wikipedia.org/wiki/ISO/IEC_8859-1)

<sup>2</sup>Codificación UTF-8 <https://en.wikipedia.org/wiki/UTF-8>

Código C++	Datos introducidos desde el teclado
<pre>#include &lt;iostream&gt; using namespace std; int main() {     int i;     double d;     char c[64];      cin &gt;&gt; i;     cin &gt;&gt; d;     cin &gt;&gt; c;     return 0; }</pre>	  10   3.14   Pepe
Código C++	Datos en el fichero datos.txt
<pre>#include &lt;iostream&gt; #include &lt;fstream&gt; using namespace std; int main() {     int i;     double d;     char c[64];     ifstream fentrada;      fentrada.open("datos.txt");     fentrada &gt;&gt; i;     fentrada &gt;&gt; d;     fentrada &gt;&gt; c;     fentrada.close();     return 0; }</pre>	  10   3.14   Pepe

Cuadro 1: Dos programas y sus datos asociados, que leen exactamente los mismos valores para cada variable.


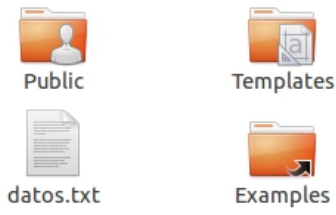
2. Crear un flujo de datos de entrada `ifstream` que proviene de un fichero cuyo nombre es `fentrada` y asociarlo al fichero `datos.txt` del que se van a leer los datos.
3. Leer los datos con el operador de extracción `>>` exactamente igual que si se leyese desde `cin`.
4. Cuando hemos terminado de leer los datos cerramos el flujo de entrada desde el fichero.

### 3.2. Procedimiento para escribir datos en un fichero

Para escribir datos en un fichero el procedimiento es muy similar y aparece ilustrado en los ejemplos del Cuadro 2.

En este caso, el procedimiento para guardar o escribir los datos en un fichero también es muy sencillo y consiste en los siguientes pasos:

1. Incluir el fichero de cabeceras `fstream` para poder manejar ficheros.

Código C++	Datos visualizados en pantalla
<pre>#include &lt;iostream&gt; using namespace std; int main() {     int i=10;     double d=3.14;     char c[64]="Pepe";      cout &lt;&lt; i &lt;&lt; " ";     cout &lt;&lt; d &lt;&lt; " ";     cout &lt;&lt; c &lt;&lt; endl;     return 0; }</pre>	 10 3.14 Pepe
Código C++	Datos guardados en el fichero datos.txt
<pre>#include &lt;iostream&gt; #include &lt;fstream&gt; using namespace std; int main() {     int i=10;     double d=3.14;     char c[64]="Pepe";     ofstream fsalida;      fsalida.open("datos.txt");     fsalida &lt;&lt; i &lt;&lt; " ";     fsalida &lt;&lt; d &lt;&lt; " ";     fsalida &lt;&lt; c &lt;&lt; endl;     fsalida.close();     return 0; }</pre>	 10 3.14 Pepe

Cuadro 2: Dos programas que escriben exactamente los mismos valores para cada variable en la pantalla (el primero) y en el fichero "datos.txt"(el segundo).

2. Crear un flujo de datos de salida `ofstream` hacia un fichero cuyo nombre es `fsalida` y asociarlo al fichero `datos.txt` en el que se van a escribir los datos. Si el fichero no existe, se crea, y si ya existe, se borran sus datos antes de empezar a escribir en él.
3. Escribir los datos en el fichero con el operador de inserción `<<` exactamente igual que si se mostrasen por pantalla con `cout`.
4. Cuando hemos terminado de escribir los datos cerramos el flujo de salida.

### 3.3. Gestión básica de errores con ficheros

Son muchas las casuísticas que se pueden presentar cuando se manejan datos desde o hacia un fichero. En esta práctica introductoria se van a comprobar los posibles errores en la apertura y en el flujo de lectura / escritura de un fichero. El primer error se produce cuando se intenta abrir un fichero que no existe o cuando no se tienen privilegios para abrir el fichero o para escribir en él. El segundo error puede ocurrir cuando las lecturas / escrituras no se han realizado con éxito. A continuación se muestra un ejemplo de la comprobación de errores asociados a las operaciones con ficheros:

```
#include<iostream>
#include<fstream>
using namespace std;

int main(){
    int i;
    double d;
    char c[64];
    ifstream fentrada;
    fentrada.open("datos.txt");
    if (fentrada){
        fentrada >> i;
        fentrada >> d;
        fentrada >> c;
        if (!fentrada){
            cerr << "error_de_lectura_del_fichero\n";
        }
        fentrada.close();
    }
    else{
        cerr << "error_de_apertura_del_fichero\n";
    }
}
```



## 4. Paso de parámetros a Main desde la línea de órdenes

Para poder pasar parámetros de cualquier tipo y en cualquier número a un programa desde la línea de comandos, la función **main** debe pasar de esta forma

```
int main() {
```

a esta otra

```
int main(int nargs, char * args[]) {
```

De esta forma, todos los parámetros indicados en la línea de órdenes pasan a la función **main()** como un vector de cadenas-c, luego, si es necesario, se pueden convertir a otros tipos de datos, por ejemplo numéricos<sup>3</sup>). Cada componente del vector es una cadena-c que contiene el respectivo parámetro de la línea de órdenes, el primer componente del vector de parámetros es el nombre del binario que se ejecuta. El número total de parámetros viene indicado como el primer parámetro de main y debe ser de tipo **int**. Así una llamada como esta

```
$$> mi_binario 10 3.14 Pepe
```

Sería recibida en el main como

```
nargs = 4
args[0] = ``mi_binario``
args[1] = ``10``
args[2] = ``3.14``
args[3] = ``Pepe``
```

<sup>3</sup>Manual de referencia de cstdlib para conversión de tipos entre cadenas-c y diferentes tipos numéricos: <http://www.cplusplus.com/reference/cstdlib/>

## 5. Práctica a entregar: gestionar idiomas

Un fichero de idiomas es un fichero que contiene una lista de bigramas identificados como frecuentes en un determinado idioma. El fichero de idiomas tiene el siguiente formato (los que se proporcionan en esta práctica están codificados en ISO8859-1)

- La primera línea contiene siempre la cadena “MP-BIGRAMAS\_IDIOMA-T-1.0”
- La segunda línea contiene una cadena que describe qué idioma es
- La tercera línea contiene el número de bigramas identificados en este fichero que están asociados al idioma descrito.
- Las siguientes líneas contienen la lista de bigramas, con sus frecuencias, según el número de bigramas especificados.

```
MP-BIGRAMAS_IDIOMA-T-1.0
spanish
548
ue 36957
de 34013
en 33672
es 32637
qu 32470
er 27172
os 25527
la 23061
do 21590
... hasta 548 bigramas en total ...
```

En esta práctica se deberá crear un programa cuya llamada sea ésta

```
idioma <bigrama> <fich1.bgr> <fich2.bgr> ... <fichn.bgr>
```

1. Todos los parámetros se pasan al programa desde la línea de órdenes.
2. El programa debe de almacenar el contenido de un fichero de idioma en memoria dinámica, tal y como se describe en la clase Idioma contenida en el fichero **Idioma.h** y que se muestra simplificado (sin comentarios) en la Sección 6.1. Esta clase hace uso de la clase Bigrama implementada en la práctica anterior.
3. El programa recibe al menos un fichero de idioma, con extensión **bgr**, con el formato descrito anteriormente. Cada fichero indicado debe estar asociado a un mismo idioma, aunque los bigramas que contengan puedan ser diferentes de un fichero a otro, por ejemplo, porque se han obtenido desde distintas fuentes.
4. El programa debe leer todos y cada uno de los ficheros de idioma indicados en la línea de órdenes y fusionarlo en un único idioma según el siguiente procedimiento.

- a) Si el bigrama leído de un fichero nuevo ya existe en el idioma que hay en memoria, se suman las dos frecuencias.
  - b) Si el bigrama nuevo no existe en el idioma que hay en memoria, se redimensiona la memoria dinámica y lo añade al final.
5. El programa debe recibir también desde la línea de órdenes, y siempre en primer lugar, un bigrama determinado. Debe buscar el bigrama en el idioma resultante de fusionar todos los ficheros de idiomas y dar la frecuencia de éste en la fusión. Como se ha indicado anteriormente, ésta frecuencia fusionada debe coincidir con la suma de las frecuencias parciales en cada fichero de idioma indicado en la llamada.
6. Se deben gestionar los errores en el acceso a los ficheros de idioma, ante cualquier error, el programa debe abortar, y por tanto liberar la memoria que pudiese haber asignada, en cualquiera de las siguientes situaciones:
- a) Que el fichero no exista
  - b) Que el fichero existe pero el idioma no coincide
  - c) Que el fichero existe, el idioma coincide pero le faltan datos

## 5.1. Corrección de la práctica

En el fichero de proyecto de NetBeans dispuesto para descarga en DECSAI, cuyo nombre es **idioma.nb.zip** se incluye un makefile con cuatro casos de prueba, cuya ejecución se muestra a continuación.

```
/usr/bin/make -f Makefile tests
Test 1
dist/Debug/GNU-Linux/idioma re data/1french.bgr data/2french.bgr

Abriendo fichero data/1french.bgr
Idioma detectado: french
Leyendo 440 bigramas
OK

Abriendo fichero data/2french.bgr
Idioma detectado: french
Leyendo 604 bigramas
OK

El bigrama re tiene una frecuencia de 9352 en el idioma french (608 bigramas)

Test 2
dist/Debug/GNU-Linux/idioma re data/1russian.bgr data/2russian.bgr

Abriendo fichero data/1russian.bgr
Idioma detectado: russian
Leyendo 462 bigramas
ERROR cargando datos del fichero data/1russian.bgr

Test 3
dist/Debug/GNU-Linux/idioma re data/1italian.bgr data/2italian.bgr

Abriendo fichero data/1italian.bgr
ERROR abriendo fichero data/1italian.bgr

Test 4
dist/Debug/GNU-Linux/idioma re data/1spanish.bgr data/2french.bgr

Abriendo fichero data/1spanish.bgr
Idioma detectado: spanish
Leyendo 548 bigramas
OK

Abriendo fichero data/2french.bgr
Idioma detectado: french
ERROR Idioma no compatible con spanish
```

Se debe asegurar con **Valgrind** que el uso de memoria dinámica es correcto.

## 6. Apéndice 1. Código de ejemplo

### 6.1. idioma.h

```
#ifndef IDIOMA.H
#define IDIOMA.H

#include "Bigrama.h"

class Idioma {
public:
    Idioma();
    Idioma(int nbg);
    void reservarMemoria(int n);
    void ampliarMemoria(int n);
    void liberarMemoria();
    std::string getIdioma() const;
    void setIdioma(const std::string& id);
    Bigrama getPosicion(int p) const;
    void setPosicion(int p, const Bigrama & bg);
    inline int getSize() const { return _nBigramas; };
    int findBigrama(const std::string bg) const;
    void addBigrama(const Bigrama & bg);
    bool cargarDeFichero(const char *fichero);
    bool addDeFichero(const char *fichero);

private:
    std::string _idioma;        /// Identificador del idioma
    Bigrama* _conjunto;        /// Vector dinámico de bigramas
    int _nBigramas;            /// Número de bigramas en el vector dinámico
};
#endif
```