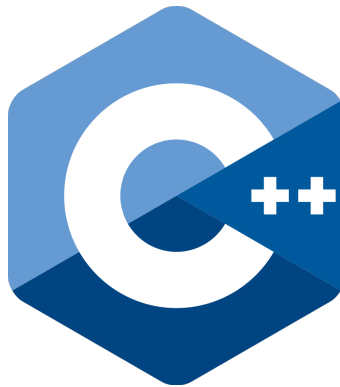




# Metodología de la Programación

DGIM, GI-C y DGADE

Curso 2022/2023



## Guion de prácticas

*Ficheros de texto*

*Marzo de 2023*



# Contents

<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>10</b>
4.1 Validando el paso de argumentos a main() . . . . .	11
4.2 Un caso especial . . . . .	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 ([Abrir →](#))

<sup>2</sup>Codificación UTF-8 ([Abrir →](#))


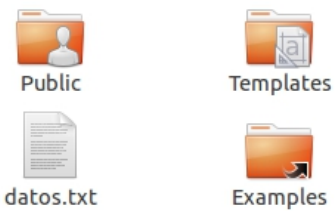
Código C++	Datos introducidos desde el teclado
<pre>#include &lt;iostream&gt; using namespace std; int main() {     int i;     double d;     char c[64];      cin &gt;&gt; i;     cin &gt;&gt; d;     cin &gt;&gt; c;     return 0; }</pre>	 10   3.14   Pepe
Código C++	Datos en el fichero datos.txt
<pre>#include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

Tabla 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.


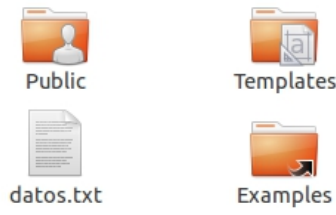
Código C++	Datos visualizados en pantalla
<pre>#include &lt;iostream&gt; using namespace std; int main() {     int i=10;     double d=3.14;     char c[64]="Pepe";      cout &lt;&lt; i &lt;&lt; " ";     cout &lt;&lt; d &lt;&lt; " ";     cout &lt;&lt; c &lt;&lt; endl;     return 0; }</pre>	 <p>10 3.14 Pepe</p>
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>	 <p>10 3.14 Pepe</p>

Tabla 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. Se deben 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. Aunque la apertura y lectura/escritura de datos haya sido la correcta, se pueden producir errores adicionales en la interpretación de los datos leídos en casos de incompatibilidad de datos o de cantidad excesiva de datos. A continuación se muestra un ejemplo de la comprobación de errores asociados a las operaciones con ficheros:

Sea el programa, leer que se muestra a continuación:

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

int main() {
    int i;
    double d;
    char c[64];
    char nombreFile[30] = "datos.txt";
    ifstream fentrada; // fichero para lectura
    fentrada.open(nombreFile);
    if (fentrada) { // comprueba estado de fentrada OK si != 0
        fentrada >> i;
        fentrada >> d;
        fentrada >> c;
        if (!fentrada) { // 2) comprueba estado. faltan datos?
            throw std::ios_base::failure(
                string("error_de_lectura_del_fichero\n")); // 1)
        }
        fentrada.close();
    }
    else { // 1)
        throw std::ios_base::failure(
            string("error_de_apertura_del_fichero_") + nombreFile);
    }
}
```

El programa lanza la excepción `std::ios_base::failure` cuando, 1) no se puede abrir el fichero o bien 2) hay algún problema durante su lectura.

La situación 1) se produce tras abrir el fichero. El estado del objeto `fentrada` no es correcto, no se ha podido abrir el fichero bien porque no existe o porque no se tiene permiso de lectura.

La situación 2) se produce tras una lectura que no ha podido ser atendida. El estado no es correcto debido a un error de lectura o alcanzar la marca de final del fichero (EOF) cuando no procede.





Dado el fichero `datos.txt` siguiente:

```
1 5 3.14 dondedigodigodigoDiego
```

La lectura es correcta. No se lanza ninguna excepción.

Si renombramos `datos.txt` a `datos.tex` desde el SO, el anterior programa leer no lo puede encontrar. Se lanza la excepción (1):

```
terminate called after throwing an instance of 'std::ios_base::failure[abi:cxx11]':  
  what(): error de apertura del fichero datos.txt: iostream error
```

Por otro lado, dado el fichero `datos.txt` siguiente:

```
1 5 3.14
```

o bien con el contenido siguiente:

```
1 5 dondedigodigodigoDiego 3.14
```

en ambos casos se lanza la excepción (2):

```
terminate called after throwing an instance of 'std::ios_base::failure[abi:cxx11]':  
  what(): error de lectura del fichero  
: iostream error
```

## 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 narg, 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

```
narg = 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: ([Abrir →](#))

## 4.1 Validando el paso de argumentos a main()

Se deben comprobar cuando los argumentos que se pasan son correctos. Vease el siguiente programa, para el que el número de argumentos es siempre el mismo.

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

void help() {
    cout << endl << "-n<number1><number2> Positive numbers to calculate product" << endl
        << "-b|h|d|o base-h-exadecimal, d-decimal, o-ctal" << endl;
}

void errorArguments() {
    cerr << "Error in arguments" << endl;
    help();
    exit(1);
}

int main(int narg, char * args[]) {
    int n=-1, m=-1, res;
    char base='0';
    // First. Check number of arguments, if possible
    if (narg != 6)
        errorArguments();
    // Second. Get the arguments
    n = atoi(args[2]);
    m = atoi(args[3]);
    base = args[5][0];
    // Third check for bad values
    if (n<0 || m<0 || (base!='h' && base != 'd' && base != 'o'))
        errorArguments();
    // Fourth. Proceed
    res = n*m;
    cout << "Result=";
    switch (base) {
        case 'h': cout << std::hex << res << endl; break;
        case 'o': cout << std::oct << res << endl; break;
        case 'd': cout << res << endl; break;
    }
    return 0;
}
```

La siguiente versión cuando aparecen parámetros opcionales.

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

void help() {
    cout << endl << "-n<number1><number2>[-b|h|d|o]" << endl
        << "-n<number1><number2> Positive numbers to calculate product" << endl
        << "-b|h|d|o base-h-exadecimal, d-decimal, o-ctal, default is h" << endl;
}

void errorArguments() {
    cerr << "Error in arguments" << endl;
    help();
    exit(1);
}

int main(int narg, char * args[]) {
    int n=-1, m=-1, res;
    char base='h';
    // First. Check number of arguments, if possible
    if (narg != 6 && narg != 4)
        errorArguments();
    // Second. Get the arguments
    n = atoi(args[2]);
    m = atoi(args[3]);
    if (narg == 6)
        base = args[5][0];
    // Third check for bad values
    if (n<0 || m<0 || (base!='h' && base != 'd' && base != 'o'))
        errorArguments();
    // Fourth. Proceed
    res = n*m;
    cout << "Result=";
    switch (base) {
        case 'h': cout << std::hex << res << endl; break;
        case 'o': cout << std::oct << res << endl; break;
        case 'd': cout << res << endl; break;
    }
    return 0;
}
```

A continuación cuando aparecen en cualquier orden.

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

void help() {
    cout << endl << " _OPTIONS_ " << endl
         << "-n<number1><number2> _Positive_numbers_to_calculate_product" << endl
         << "-b_h|d|o_base_h-exadecimal,_d-ecimal,_o-ctal,_default-is_h" << endl;
}

void errorArguments() {
    cerr << "Error_in_arguments" << endl;
    help();
    exit(1);
}

int main(int narg, char * args[]) {
    int n=-1, m=-1, res;
    char base='h';
    // First. Check number of arguments, if possible
    if (narg != 6)
        errorArguments();
    // Second. Get the arguments
    for (int i=1; i<narg; i++) {
        string sarg=args[i++];
        if (sarg == "-n") {
            n = atoi(args[i++]);
            m = atoi(args[i++]);
        } else if (sarg == "-b") {
            base = args[i++][0];
        } else
            errorArguments();
    }
    // Third check for bad values
    if (n<0 || m<0 || (base!='h' && base != 'd' && base != 'o'))
        errorArguments();
    // Fourth. Proceed
    res = n*m;
    cout << "Result=_";
    switch (base) {
        case 'h': cout << std::hex << res << endl; break;
        case 'o': cout << std::oct << res << endl; break;
        case 'd': cout << res << endl; break;
    }
    return 0;
}
```

Y, finalmente, cuando aparecen un número no determinado de parámetros.

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

void help() {
    cout << endl << "-b_h|d|o _n _n<number1><number2>[_<number1><number2>..._]" << endl
         << "-n<number1><number2> _Positive_numbers_to_calculate_product" << endl
         << "-b_h|d|o_base_h-exadecimal,_d-ecimal,_o-ctal,_default-is_h" << endl;
}

void errorArguments() {
    cerr << "Error_in_arguments" << endl;
    help();
    exit(1);
}

int main(int narg, char * args[]) {
    int n=-1, m=-1, res;
    char base='h';
    // First. Check number of arguments, if possible
    if (narg < 3 || narg%2 == 1)
        errorArguments();
    // Second. Get the arguments
    base = args[2][0];

    // Third check for bad values
    if (base!='h' && base != 'd' && base != 'o')
        errorArguments();
    // Fourth. Proceed while processing remaining arguments
    for (int i=4; i<narg; i+=2) {
        n = atoi(args[i]);
        m = atoi(args[i+1]);
        // Fifth check arguments while processing them
        if (n<0 || m<0)
            errorArguments();
        res = n*m;
        cout << std::dec << n << "x" << m << "=_";
        switch (base) {
            case 'h': cout << std::hex << res << endl; break;
            case 'o': cout << std::oct << res << endl; break;
            case 'd': cout << std::dec << res << endl; break;
        }
    }
    return 0;
}
```

## 4.2 Un caso especial

A veces se pide leer (escribir) los datos desde teclado o desde un fichero indistintamente según un parámetro de main().

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

void help() {
    cout << endl << "[_OPTIONS_]" << endl
    << "-i<filename> _Input_data_from_this_file ,_otherwise_from_keyboard" << endl
    << "-b<h|d|o> _base_h-exadecimal ,_d-decimal ,_o-octal ,_default_is_h" << endl;
}

void errorArguments() {
    cerr << "Error in arguments" << endl;
    help();
    exit(1);
}

int main(int narg, char * args[]) {
    int n=-1, m=-1, res;
    char base='h';
    string filename="";
    ifstream ifile;
    istream *input=&cin;

    // First. Check number of arguments, if possible

    // Second. Get the arguments
    for (int i=1; i<narg; i++) {
        string sarg=args[i++];
        if (sarg == "-i") {
            filename = args[i++];
            ifile.open(filename.c_str());
            if (!ifile) {
                cerr << "Error opening file " << filename << endl;
                exit(1);
            }
            input = &ifile;
        } else if (sarg=="-b") {
            base = args[i++][0];
        } else
            errorArguments();
    }

    // Third check for bad values
    if (base!='h' && base != 'd' && base != 'o')
        errorArguments();

    // Fourth. Proceed
    (*input) >> n >> m;
    if (input->eof()) {
        cerr << "Error reading data from " << filename << endl;
        exit(1);
    }

    if (n<0 || m<0)
        errorArguments();

    res = n*m;
    cout << "Result=";
    switch (base) {
        case 'h': cout << std::hex<<res<<endl; break;
        case 'o': cout << std::oct<<res<<endl; break;
        case 'd': cout << std::dec<<res<<endl; break;
    }

    if (input != &cin)
        ifile.close();

    return 0;
}
```