

Programación 1

Tema 13

Ficheros



Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza





Información sobre protección de datos de carácter personal en el tratamiento de gestión de grabaciones de docencia

Sesión con grabación



Tratamiento: Gestión de grabaciones de docencia

Finalidad: Grabación y tratamiento audiovisual de docencia y su evaluación

Base Jurídica: Art. 6.1.b), c) y d) Reglamento General de Protección de Datos

Responsable: Universidad de Zaragoza.

Ejercicio de Derechos de acceso, rectificación, supresión, portabilidad, limitación u oposición al tratamiento ante el gerente de la Universidad conforme a <https://protecciondatos.unizar.es/procedimiento-seguir>

Información completa en:

https://protecciondatos.unizar.es/sites/protecciondatos.unizar.es/files/user/s/lopd/gdocencia_extensa.pdf

Propiedad intelectual: Queda prohibida la difusión, distribución o divulgación de la grabación y particularmente su compartición en redes sociales o servicios dedicados a compartir apuntes. La infracción de esta prohibición puede generar responsabilidad disciplinaria, administrativa y de índole civil o penal.

Fuente de las imágenes: <https://pixabay.com/es>



Información sobre protección de datos de carácter personal en el tratamiento de gestión de grabaciones de docencia

- Se recuerda que la grabación de las clases por medios distintos a los usados por el profesor o por personas diferentes al profesor sin su autorización expresa no está permitida, al igual que la difusión de esas imágenes o audios.

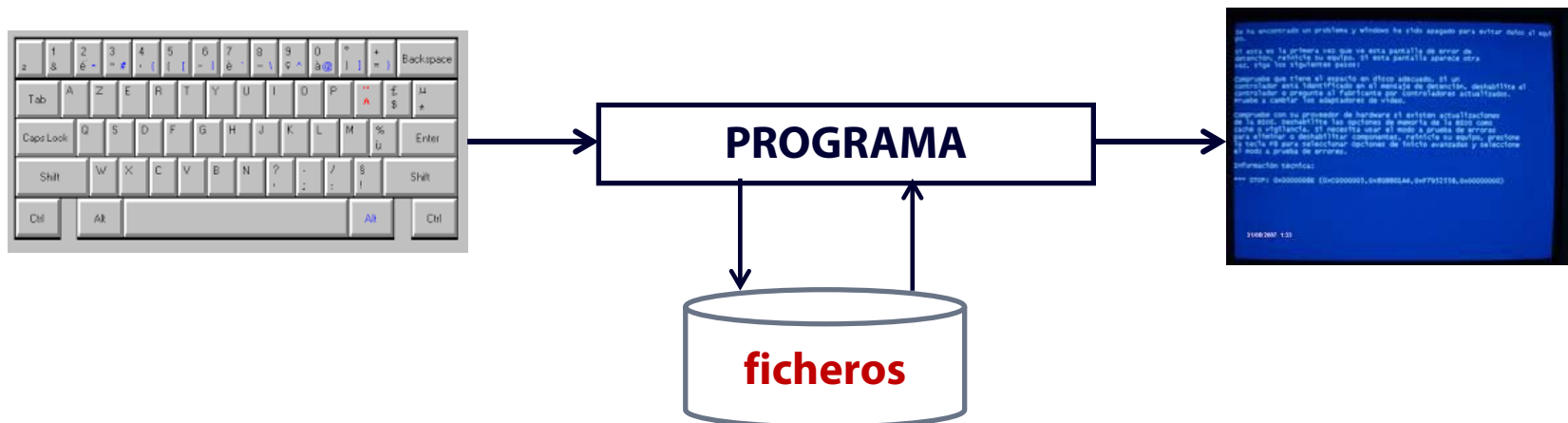
Objetivos

- ❑ Interacción de un programa con su entorno (consola, sistema de ficheros) leyendo o escribiendo datos
- ❑ Fichero como secuencia persistente de datos
- ❑ Herramientas de C++ para entrada y salida de datos

Entrada y salida (E/S) de datos

- Un programa necesita datos del entorno y proporciona información y resultados al entorno:
 - Leyendo datos del teclado
 - Escribiendo o presentando datos en la pantalla
 - Leyendo datos de ficheros
 - Escribiendo o almacenando datos en ficheros

Entrada y salida (E/S) de datos

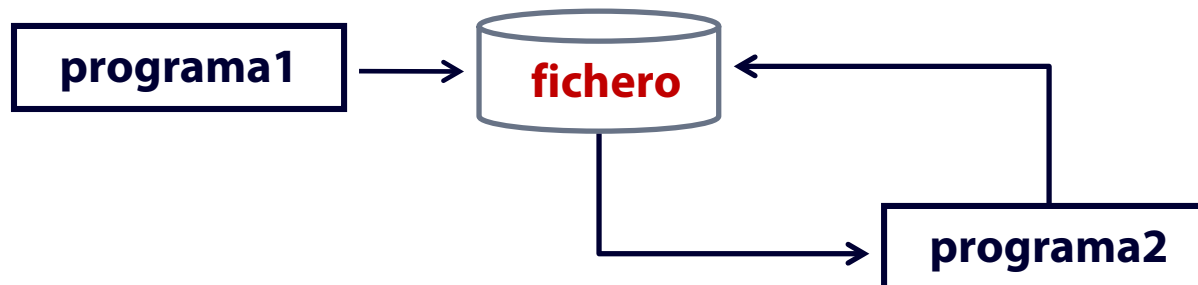


Ficheros o archivos de datos

- Un **fichero** o **archivo** almacena una **secuencia de *bytes***, ilimitada pero finita:
 - $\langle b_1, b_2, b_3, \dots, b_k \rangle$
 - La capacidad de un fichero o archivo no está limitada a priori.
 - El contenido **de todos** los ficheros puede verse como una secuencia de *bytes* (datos de tipo **char** en C++).

Ficheros o archivos de datos

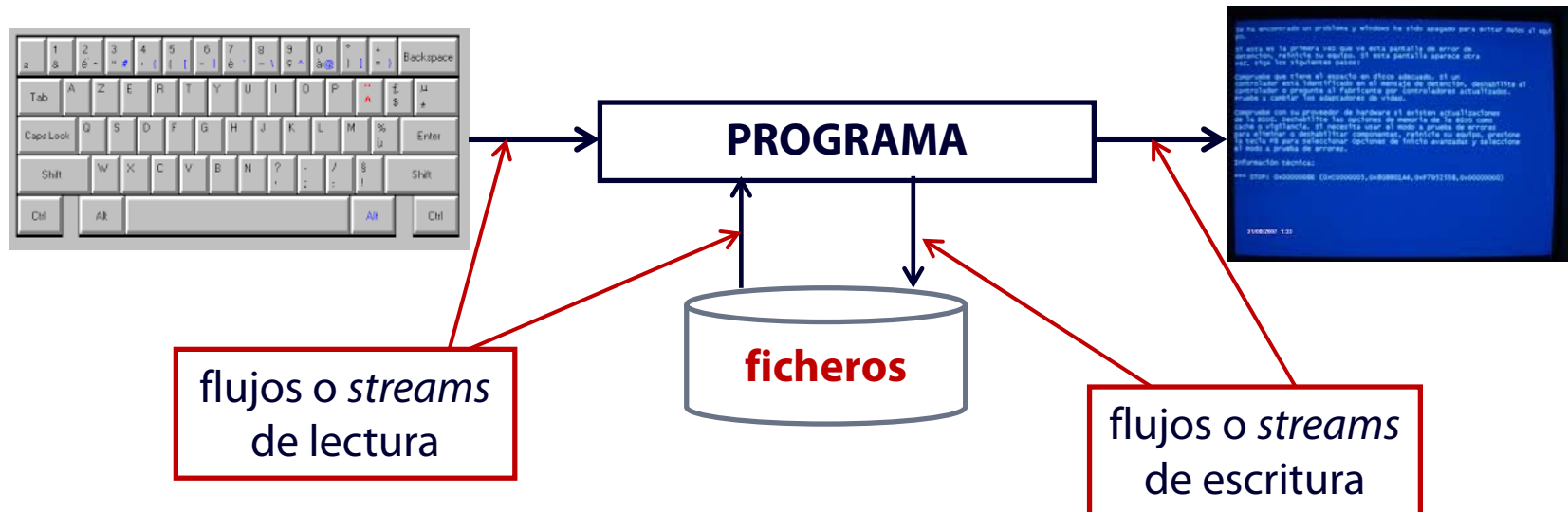
- Los datos de un fichero o archivo son **persistentes**:
 - Sobreviven a la ejecución del programa y puedan utilizarse posteriormente.



Flujos

- La comunicación de datos entre un programa C++ y su entorno se fundamenta en el concepto de **flujos** o ***streams***
 - Comunican información entre un origen y un destino
 - El programa C++ es uno de los extremos del flujo (el destino o el origen de la información)
 - El otro extremo del flujo puede ser
 - un dispositivo físico (teclado, pantalla)
 - un fichero almacenado en un dispositivo físico
 - La comunicación se produce
 - Leyendo *byte a byte* del flujo
 - Escribiendo *byte a byte* en el flujo

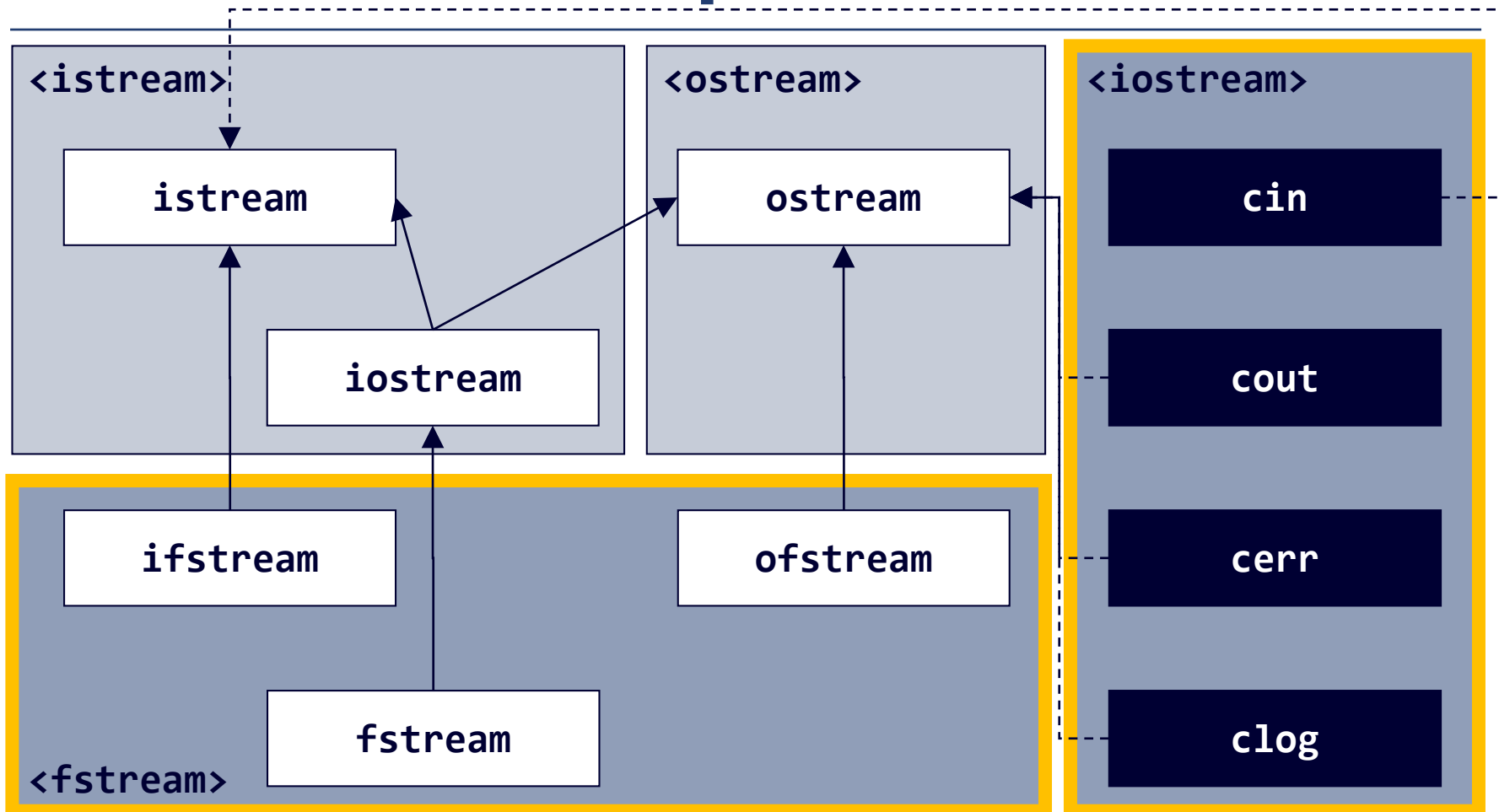
Flujos



Herramientas C++ para E/S

- Biblioteca **<iostream>**
 - Flujos de entrada de la clase **istream** para la lectura de datos
- Biblioteca **<ostream>**
 - Flujos de salida de la clase **ostream** para la escritura de datos
- Biblioteca **<iostream>**
 - Flujos predefinidos **cin** de la clase **istream** y **cout**, **cerr** y **clog** de la clase **ostream**
- Biblioteca **<fstream>**
 - Flujos de entrada de la clase **ifstream** para la lectura de ficheros
 - Flujos de salida de la clase **ofstream** para la escritura de ficheros
 - Flujos de entrada y salida de la clase **fstream** para la lectura y escritura de ficheros

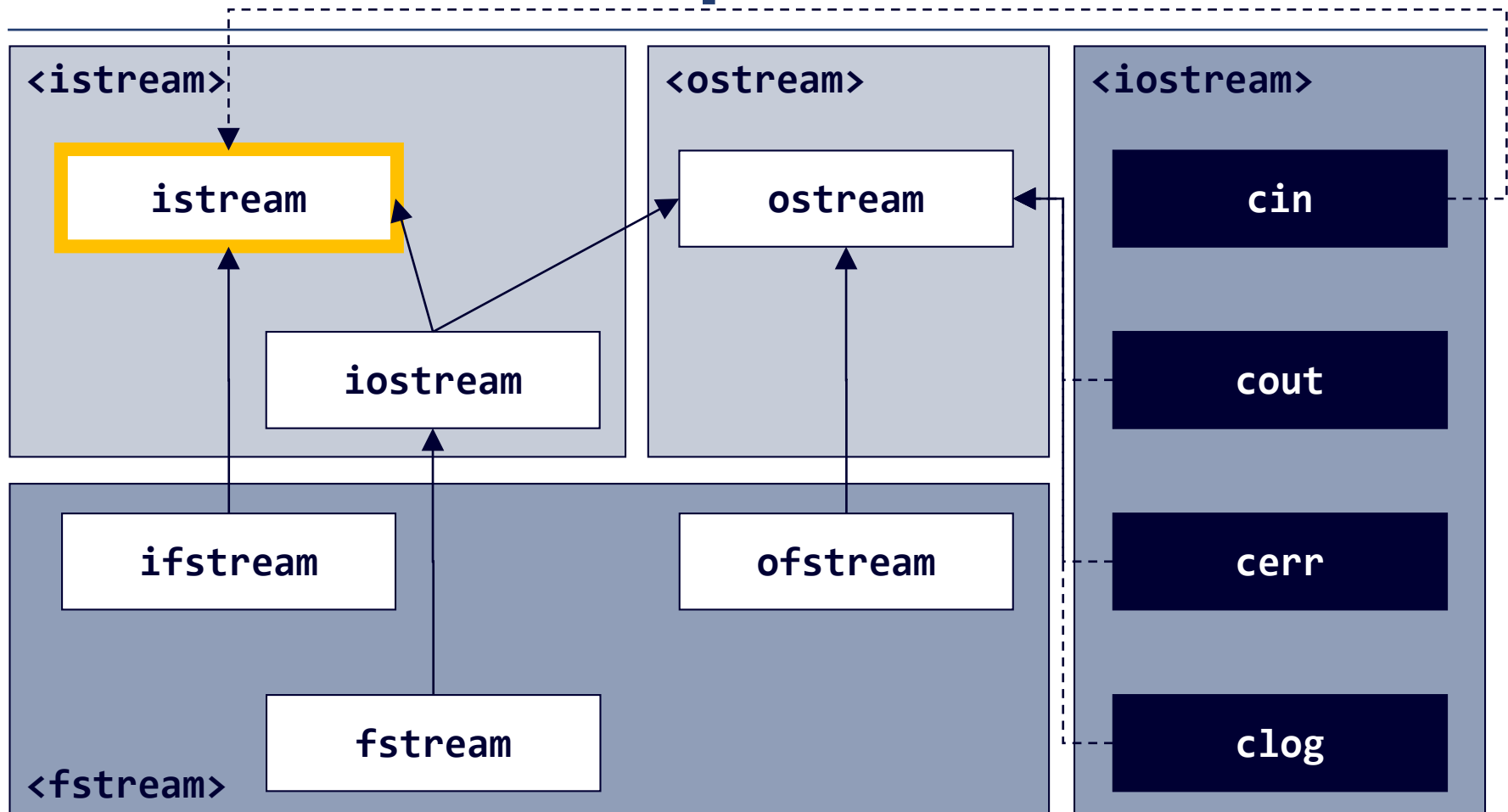
Herramientas C++ para E/S



Biblioteca <iostream>

- Ofrece cuatro objetos para gestionar cuatro flujos predefinidos
 - **cin**
 - Objeto de la clase **istream**
 - Gestiona el flujo de entrada estándar (entrada de datos desde teclado)
 - **cout**
 - Objeto de la clase **ostream**
 - Gestiona el flujo de salida estándar (presentación de datos en la pantalla)
 - **cerr**
 - Objeto de la clase **ostream**
 - Gestiona el flujo de salida de mensajes de error (por defecto, en la pantalla)
 - **clog**
 - Objeto de la clase **ostream**
 - Gestiona el flujo de salida de mensajes de historial o registros, *log*, (por defecto, en la pantalla)

Herramientas C++ para E/S



Clase `istream`

- Definida en la biblioteca `<istream>`
- Métodos básicos, dado un objeto `f` de la clase **`istream`**
 - `f.get()`
 - Extrae el siguiente *byte* pendiente de leer del flujo de entrada `f` y lo devuelve como resultado.
 - `f.get(char& c)`
 - Extrae el siguiente *byte* pendiente de leer del flujo en entrada `f` y lo asigna al parámetro `c`.

Ejemplo

- Si en el flujo de entrada *f* se ha escrito:

Prog 1
- Las invocaciones a *f.get()* del siguiente código devuelven lo siguiente:

```
char c = f.get();      // «c» sería igual a 'P'  
c = f.get();           // «c» sería igual a 'r'  
c = f.get();           // «c» sería igual a 'o'  
c = f.get();           // «c» sería igual a 'g'  
c = f.get();           // «c» sería igual a ' '  
c = f.get();           // «c» sería igual a '1'
```


Ejemplo

- El flujo de entrada podría ser el teclado:

Prog 1

- Las invocaciones a `cin.get()` del siguiente código devolverían lo siguiente:

```
char c = cin.get();    // «c» sería igual a 'P'  
c = cin.get();        // «c» sería igual a 'r'  
c = cin.get();        // «c» sería igual a 'o'  
c = cin.get();        // «c» sería igual a 'g'  
c = cin.get();        // «c» sería igual a ' '  
c = cin.get();        // «c» sería igual a '1'
```

Funciones de la biblioteca `<string>` para trabajar con objetos `istream`

- `getline(istream& f, string& cadena);`
 - Extrae una cadena de caracteres del flujo de entrada `f`. Extrae caracteres hasta que se encuentra con el carácter `'\n'`, que extrae también del flujo `f`. Asigna a la cadena de caracteres `cadena` los caracteres extraídos, excepto el carácter `'\n'`.
- `getline(istream& f, string& cadena, char delimitador);`
 - Extrae una cadena de caracteres del flujo de entrada `f`. Extrae caracteres hasta que se encuentra con el carácter `delimitador`, que extrae también del flujo `f`. Asigna a la cadena de caracteres `cadena` los caracteres extraídos, excepto el carácter `delimitador`.

Ejemplo.


Función getline

```
int main() {  
    cout << "Escriba una línea: ";  
    string lineaCompleta;  
    getline(cin, lineaCompleta);  
    cout << "La línea escrita era \"  
        << lineaCompleta  
        << "\".\" << endl;  
    return 0;  
}
```

Ejemplo.

Función getline con delimitador

```
int main() {  
    cout << "Escriba otra línea: ";  
    string trozoDeLinea;  
    getline(cin, trozoDeLinea, 'e');  
    cout << "La línea escrita hasta la "  
        << "primera 'e' era \""  
        << trozoDeLinea << "\"." << endl;  
    getline(cin, trozoDeLinea);  
    cout << "El resto de la línea era \""  
        << trozoDeLinea << "\"." << endl;  
    return 0;  
}
```





Operaciones de la clase **istream** para lectura con formato

- Operador de extracción `>>` para la lectura de una secuencia de datos a través de un flujo de entrada:
 - Ejemplo: `cin >> v1 >> v2 >> v3;`
 - Disponible para:
 - `char`
 - `int / unsigned int`
 - `double`
 - `bool`
 - `string`

Operador >> de extracción de un flujo para enteros

- `cin >> variableEntera;`
- `f >> variableEntera;`
- Se extraen del flujo todos los caracteres blancos que haya (espacios en blanco ' ', tabuladores '\t' y finales de línea '\n'), que se ignoran.
- Se extraen del flujo todos los caracteres que forman parte de la sintaxis de un entero válido (signo '+' o '-' y dígitos), que se procesan para dar valor a `variableEntera`.
- El primer carácter pendiente de leer del flujo es el que sigue al último dígito extraído del flujo.

Operador >> de extracción de un flujo para reales

- `cin >> variableReal;`
- `f >> variableReal;`
- Se extraen del flujo todos los caracteres blancos que haya (espacios en blanco ' ', tabuladores '\t' y finales de línea '\n'), que se ignoran.
- Se extraen del flujo todos los caracteres que forman parte de la sintaxis de un entero válido (signo '+' o '-', dígitos y '.'), que se procesan para dar valor a `variableReal`.
- El primer carácter pendiente de leer del flujo es el que sigue al último dígito (o punto decimal) extraído del flujo.

Operador >> de extracción de un flujo para caracteres

- `cin >> variableCaracter;`
- `f >> variableCaracter;`
- Se extraen del flujo todos los caracteres blancos que haya (espacios en blanco ' ', tabuladores '\t' y finales de línea '\n'), que se ignoran.
- Se extraen del flujo el primer carácter distinto a espacio en blanco, tabulador o fin de línea, que da valor a `variableCaracter`.
- El primer carácter pendiente de leer del flujo es el que sigue al último carácter extraído y que ha dado valor a `variableCaracter`.

Operador >> de extracción de un flujo para cadenas de caracteres

- `cin >> variableCadena;`
- `f >> variableCadena;`
- Se extraen del flujo todos los caracteres blancos que haya (espacios en blanco ' ', tabuladores '\t' y finales de línea '\n'), que se ignoran.
- Se extraen del flujo todos los caracteres que siguen y que son distintos a espacio en blanco, tabulador o fin de línea, que da valor a `variableCadena`.
- El primer carácter pendiente de leer del flujo es el primer carácter que sigue al último que ha dado valor a `variableCadena` y que es, por tanto, un espacio en blanco, tabulador o fin de línea.

Ejemplo.

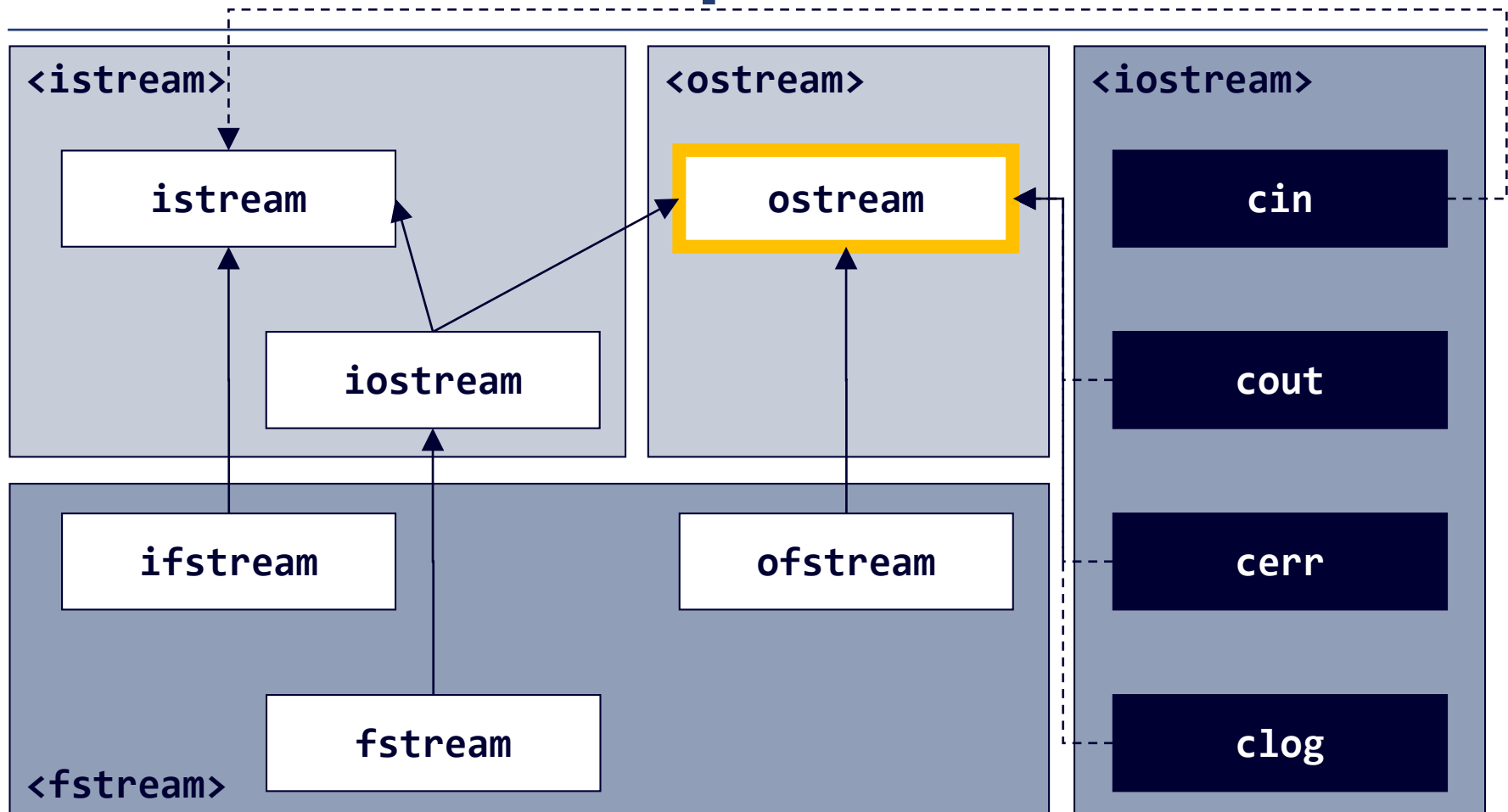
Operadores de extracción

```
int main() {  
    cout << "Escriba un entero, un real, un carácter y "  
        << "una palabra: ";  
    int entero;  
    double real;  
    char character;  
    string palabra;  
    cin >> entero >> real >> character >> palabra;  
    cout << "Los datos leídos son:" << endl;  
    cout << "Entero: " << entero << endl;  
    cout << "Real: " << real << endl;  
    cout << "Carácter: '" << character << "'" << endl;  
    cout << "Palabra: \"" << palabra << "\"" << endl;  
    return 0;  
}
```

Otro ejemplo. Operadores de extracción con otros separadores

```
int main() {  
    cout << "Escriba un entero, un real, un carácter y una "  
    cout << "palabra (separados por comas): ";  
    int entero; double real; char caracter; string palabra;  
    cin >> entero;  
    cin.get();           // Nos saltamos el carácter separador ','  
    cin >> real;  
    cin.get();           // Nos saltamos el carácter separador ','  
    cin >> caracter;  
    cin.get();           // Nos saltamos el carácter separador ','  
    cin >> palabra;  
    cout << "Los datos leídos son:" << endl;  
    ...  
    return 0;  
}
```

Herramientas C++ para E/S



Clase ostream

- Definida en la biblioteca `<ostream>`
- Método básico, dada la declaración `ostream f;`
 - `f.put(char c)`
 - Inserta el carácter `c` en el flujo de salida `f`
- Operador de inserción `<<` para la escritura de una secuencia de datos a través de un flujo de salida:
 - Ejemplo: `cout << d1 << d2 << d3;`

ostream

Ejemplo de put vs. <<

```
int main() {  
    char character = 'E';  
  
    cout << "Escritura de caracteres con put: ";  
    cout.put('a');  
    cout.put(character);  
    cout << endl;  
  
    cout << "Escritura de caracteres con <<: ";  
    cout << 'a';  
    cout << character;  
    cout << endl;  
  
    return 0;  
}
```

ostream

Ejemplo de uso del operador de inserción

```
int main() {  
    cout << "Escritura de otros tipos de datos con <<: "  
        << endl;  
    cout << -23 << endl;  
    cout << 3.1415927 << endl;  
    cout << "Cadena literal de caracteres" << endl;  
    string cadena = "Cadena de la clase string";  
    cout << cadena << endl;  
    cout << boolalpha << true << ", " << false << endl;  
    cout << noboolalpha << true << ", " << false << endl;  
    return 0;  
}
```

Biblioteca <ostream>

- Manipuladores

- **flush**

- vacía el búfer asociado al flujo de salida.

- **endl**

- Inserta el carácter de fin de línea ' \n ' en el flujo de salida y vacía el búfer asociado a dicho flujo.

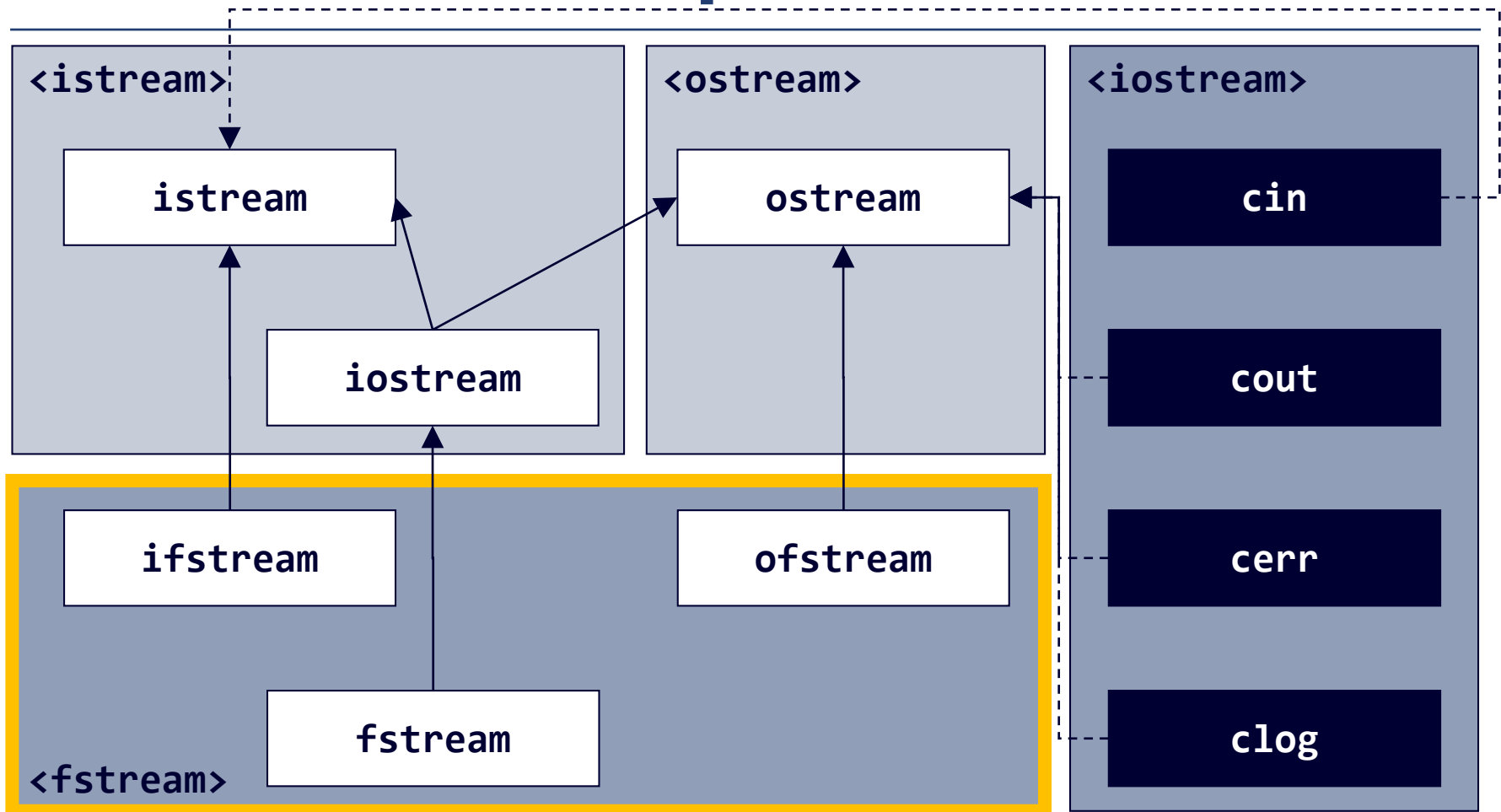
Resumen de las clases `istream` y `ostream`

- Operaciones para leer de un objeto `f` de la clase **`istream`**
 - `int f.get()`
 - `f.get(char& c)`
 - `getline(istream& f, string& str)`
 - `getline(istream& f, string& str, char delimitador)`
 - `f >> variable_char`
 - `f >> variable_int`
 - `f >> variable_double`
 - `f >> variable_string`

Resumen de las clases `istream` y `ostream`

- Operaciones para escribir en un objeto `f` de la clase **`ostream`**
 - `f.put(char c)`
 - `f << expresión_char`
 - `f << expresión_int`
 - `f << expresión_double`
 - `f << expresión_cadena`
 - `f << flush`
 - `f << endl`

Herramientas C++ para E/S



Entrada y salida de datos en ficheros

- Biblioteca predefinida **<fstream>**
 - Define tres clases para trabajar con ficheros de datos
 - **ifstream**
 - **ofstream**
 - **fstream**

Entrada y salida de datos en ficheros

- **ifstream**
 - Clase cuyos objetos permiten gestionar un **flujo de entrada asociado a un fichero** y leer sus datos
- **ofstream**
 - Clase cuyos objetos permiten gestionar un **flujo de salida asociado a un fichero** y escribir datos en él
- **fstream**
 - Clase cuyos objetos permiten gestionar un **flujo de entrada y salida asociado a un fichero** y leer datos almacenados en él y escribir nuevos datos en él

Funciones de la biblioteca <fstream>

- Operaciones para gestionar ficheros externos con un objeto `f` de las clases `ifstream` u `ofstream`:
 - `f.open(const string nombreFichero)`
 - Asocia el fichero de nombre `nombreFichero` al flujo `f`
 - `f.is_open()`
 - Devuelve `true` si y solo si el flujo `f` está asociado a un fichero
 - `f.close()`
 - Libera el fichero asociado al flujo `f` y lo disocia de este
- Operaciones adicionales para gestionar la lectura de ficheros externos con un objeto `f` de la clase `ifstream`:
 - `f.eof()`
 - devuelve `true` si y solo si la última operación de lectura no pudo completarse por no haber ya datos pendientes de lectura en el flujo `f`

Ejemplo

Escritura de datos en un fichero

```
/*  
 * Pre: ---  
 * Post: Ha creado un fichero denominado  
 * "miPrimerFichero.txt" y ha escrito en  
 * él las letras mayúsculas del alfabeto  
 * inglés. En caso de que se haya  
 * producido un error, ha informado de  
 * ello escribiendo en «cerr».  
 */  
void crearFichero();
```

Ejemplo

Escritura de datos en un fichero

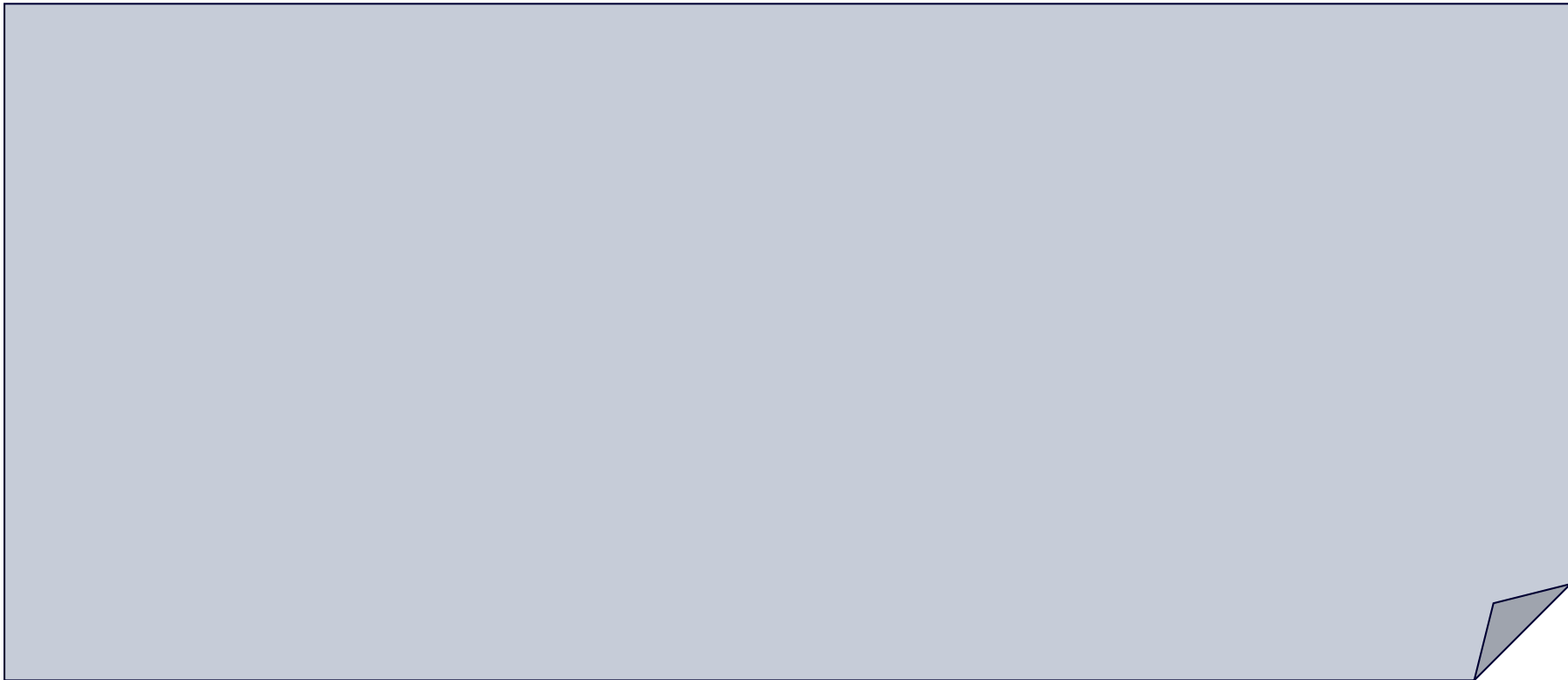
```
void crearFichero() {
    ofstream f;
    f.open("mi-primer-fichero.txt");
    if (f.is_open()) {
        for (char letra = 'A'; letra <= 'Z'; letra++) {
            f.put(letra);
        }
        f.close();
    }
    else {
        cerr << "No se ha podido crear el fichero "
              << "\"miPrimerFichero.txt\"" << endl;
    }
}
```




Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt





Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt

A



Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt

AB



Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt

ABC



Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt

ABCD



Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt

ABCDE



Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Ejemplo

Lectura de datos de un fichero

```
/*  
 * Pre: ---  
 * Post: Si «nombreFichero» define el nombre de  
 * un fichero, entonces muestra su  
 * contenido por pantalla; en caso  
 * contrario advierte del error  
 * escribiendo un mensaje por pantalla.  
 */  
void mostrar(const string nombreFichero);
```


Ejemplo

Lectura de datos de un fichero

```
void mostrar(const string nombreFichero) {  
    ifstream f;                                // Declara un flujo de entrada  
    f.open(nombreFichero);                      // Le asocia el fichero nombreFichero  
    if (f.is open()) {  
        char c = f.get();                      // Intenta leer el primer carácter  
        while (!f.eof()) {  
            cout << c;                          // Presenta el último carácter leído  
            c = f.get();                        // Intenta leer un nuevo carácter  
        }  
        f.close();                             // Disocia el fichero y lo libera  
    }  
    else {  
        cerr << "No se ha podido acceder a \""  
            << nombreFichero << "\"\" << endl;  
    }  
}
```

Ejemplo

Lectura de datos de un fichero

```
int main() {  
    mostrar("miPrimerFichero.txt");  
    return 0;  
}
```



Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt

ABCDEFGHIJKLMNOPQRSTUVWXYZ

variable c de la función mostrar:

?

pantalla



Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

variable c de la función mostrar:

'A'

pantalla



Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

variable c de la función mostrar:

'A'

pantalla

A



Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt

ABCDEFGHIJKLMNOPQRSTUVWXYZ

variable c de la función mostrar:

'B'

pantalla

A



Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt

ABCDEFGHIJKLMNOPQRSTUVWXYZ

variable c de la función mostrar:

'B'

pantalla

AB



Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt

ABCDEFGHIJKLMNOPQRSTUVWXYZ

variable c de la función mostrar:

'C'

pantalla

AB



Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt

ABCDEFGHIJKLMNOPQRSTUVWXYZ

variable c de la función mostrar:

'C'

pantalla

ABC



Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt

ABCDEFGHIJKLMNOPQRSTUVWXYZ

variable c de la función mostrar:

'D'

pantalla

ABC



Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt

ABCDEFGHIJKLMNOPQRSTUVWXYZ

variable c de la función mostrar:

'D'

pantalla

ABCD

Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt

ABCDEFGHIJKLMNOPQRSTUVWXYZ

variable c de la función mostrar:

'Y'

pantalla

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt

ABCDEFGHIJKLMNOPQRSTUVWXYZ

variable c de la función mostrar:

'Z'

pantalla

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt

ABCDEFGHIJKLMNOPQRSTUVWXYZ

variable c de la función mostrar:

'Z'

pantalla

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Ejemplo

Escritura de datos en un fichero

mi-primer-fichero.txt

ABCDEFGHIJKLMNOPQRSTUVWXYZ

variable c de la función mostrar:

xFF

Carácter de código 255 (–1 con signo).

- En CP850, corresponde con el carácter denominado «NBSP».
- En UTF-8, no se corresponde con ninguna codificación válida (xFF).

pantalla

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Ejemplo

Lectura de datos de un fichero

```
void mostrar(const string nombreFichero) {  
    ifstream f;                                // Declara un flujo de entrada  
    f.open(nombreFichero);                      // Le asocia el fichero nombreFichero  
    if (f.is open()) {  
        char c;  
        f.get(c);                                // Intenta leer el primer carácter  
        while (!f.eof()) {  
            cout << c;                            // Presenta el último carácter leído  
            f.get(c);                                // Intenta leer un nuevo carácter  
        }  
        f.close();                                // Disocia el fichero y lo libera  
    }  
    else {  
        cerr << "No se ha podido acceder a \""  
            << nombreFichero << "\"\" << endl;  
    }  
}
```


Ejemplo. Copia

```
/*  
 * Pre: ---  
 * Post: Si «nombreFichero» define el nombre de  
 * un fichero, copia su contenido en  
 * «nombreCopia»; en caso contrario o en  
 * caso de otro error, advierte del  
 * mismo escribiendo un mensaje en la  
 * pantalla.  
 */  
void copiar(const string nombreFichero,  
            const string nombreCopia);
```

Ejemplo. Copia

```
void copiar(const string nombreFichero,
            const string nombreCopia) {
    ifstream fOriginal;
    fOriginal.open(nombreFichero);
    if (fOriginal.is_open()) {
        ...
        fOriginal.close();
    }
    else {
        cerr << "No se ha podido acceder a \""
              << nombreFichero << "\"." << endl;
    }
}
```

Ejemplo. Copia

```
void copiar(const string nombreFichero,  
           const string nombreCopia) {  
    ...  
    ofstream fCopia;  
    fCopia.open(nombreCopia);  
    if (fCopia.is_open()) {  
        ...  
        fCopia.close();  
    }  
    else {  
        cerr << "No se ha podido escribir en \"" << nombreCopia  
              << "\"." << endl;  
    }  
    ...  
}
```

Ejemplo. Copia (carácter a carácter)

```
void copiar(const string nombreFichero,  
           const string nombreCopia) {  
  
    ...  
  
    char c = fOriginal.get();  
    while (!fOriginal.eof()) {  
        fCopia.put(c);  
        c = fOriginal.get();  
    }  
  
    ...  
}
```

Ejemplo. Copia (línea a línea)

```
void copiar(const string nombreFichero,  
           const string nombreCopia) {  
    ...  
    string linea;  
    getline(fOriginal, linea);  
    while (!fOriginal.eof()) {  
        fCopia << linea << endl;  
        getline(fOriginal, linea);  
    }  
    ...  
}
```

Resumen bibliotecas

<iostream>, <ostream> y <fstream>

Operaciones disponibles para leer de un objeto f de la clase ifstream	Operaciones disponibles para leer de un objeto g de la clase ofstream
Por ser un istream: <code>int f.get()</code> <code>f.get(char& c)</code> <code>getline(istream& f, string cad)</code> <code>getline(istream& f, string cad, char delimitador)</code> <code>f >> variable_char</code> <code>f >> variable_int</code> <code>f >> variable_double</code> <code>f >> variable_string</code>	Por ser un ostream: <code>g.put(char c)</code> <code>g << expresión_char</code> <code>g << expresión_int</code> <code>g << expresión_double</code> <code>g << expresión_cadena</code> <code>g << flush</code> <code>g << endl</code>
Por ser un ifstream: <code>f.open(string nombreFichero[])</code> <code>f.is_open()</code> <code>f.close()</code> <code>bool f.eof()</code>	Por ser un ofstream: <code>g.open(string nombreFichero[])</code> <code>g.is_open()</code> <code>g.close()</code>

¿Cómo se puede estudiar este tema?

- Repasando estas transparencias
- Trabajando con el código de estas transparencias
 - <https://github.com/prog1-eina/tema-13-ficheros>
- Leyendo
 - Capítulo 13 de los apuntes del profesor Martínez
 - Tutoriales de *Cplusplus.com* (2000–2017)
 - «Basic Input/Output»: http://www.cplusplus.com/doc/tutorial/basic_io/
 - «Input/output with files»: <http://www.cplusplus.com/doc/tutorial/files/>
 - En ambos casos se introducen y explican más conceptos de los que se van a ver en este curso
- Problemas de las clases de diciembre
- Práctica 6 y trabajo obligatorio.