Programación 1

Tema 13

Ficheros

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

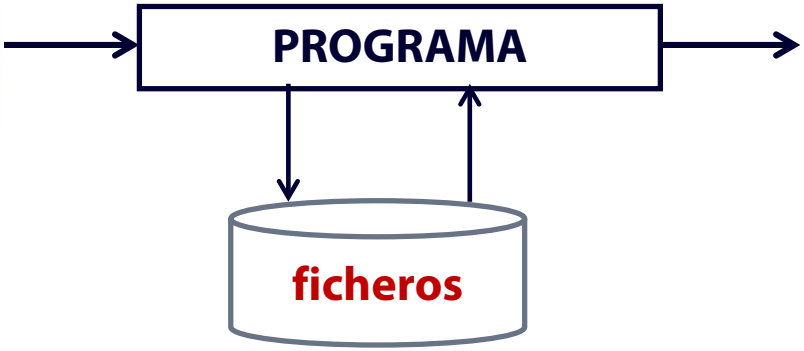
Objetivos

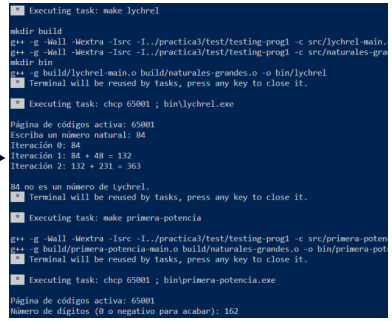
- Interacción de un programa con su entorno (terminal, sistema de ficheros) leyendo o escribiendo datos
- Fichero como secuencia persistente de datos
- Herramientas de C++ para entrada y salida de datos

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





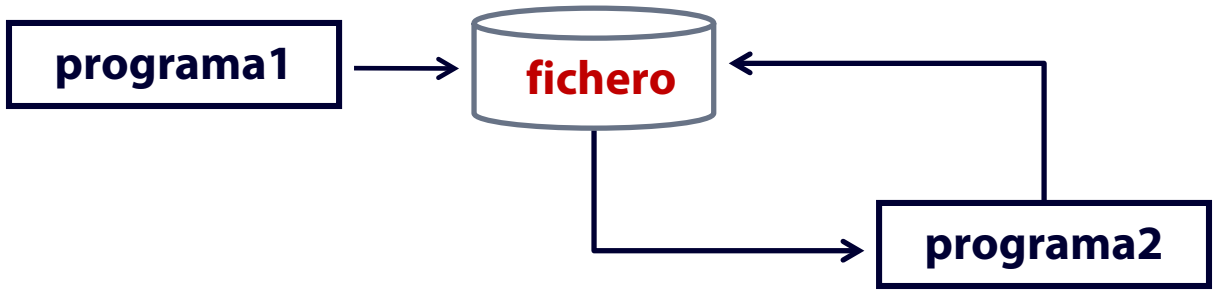
Fuente: George Hodan. «Computer Keyboard». Public Domain Pictures.net. <https://www.publicdomainpictures.net/en/view-image.php?image=164106> CC0 Public Domain,

Ficheros o archivos de datos

- Un **fichero** o **archivo** almacena una **secuencia de *bytes*,** ilimitada pero finita:
  - $<b_1, b_2, b_3, \ldots, b_k>$
  - 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++).

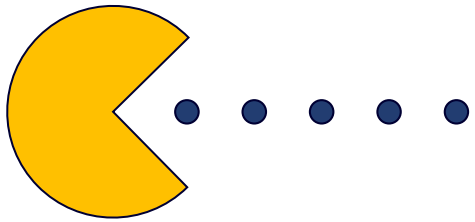
Ficheros o archivos de datos

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

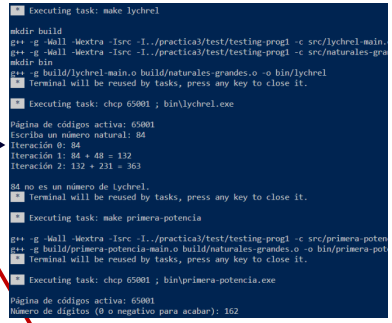


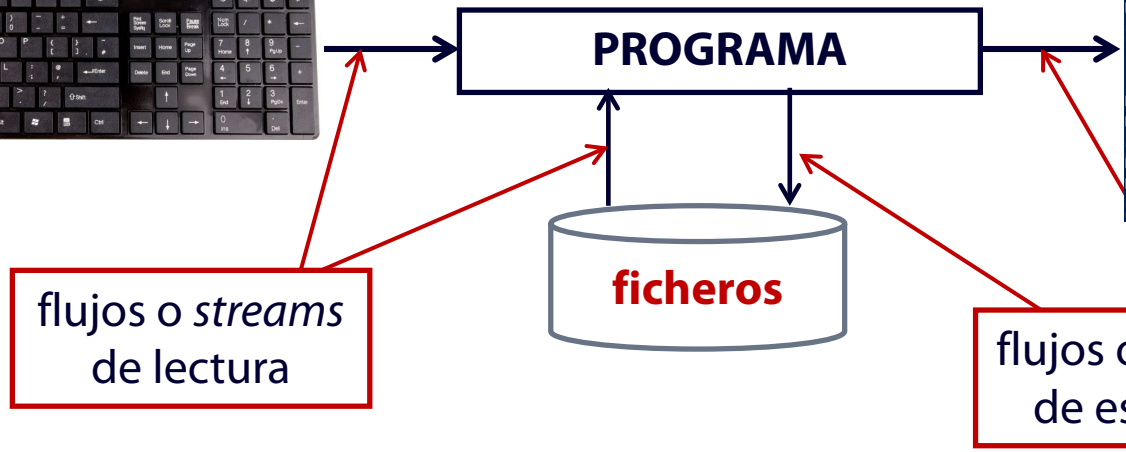
Flujos

- 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



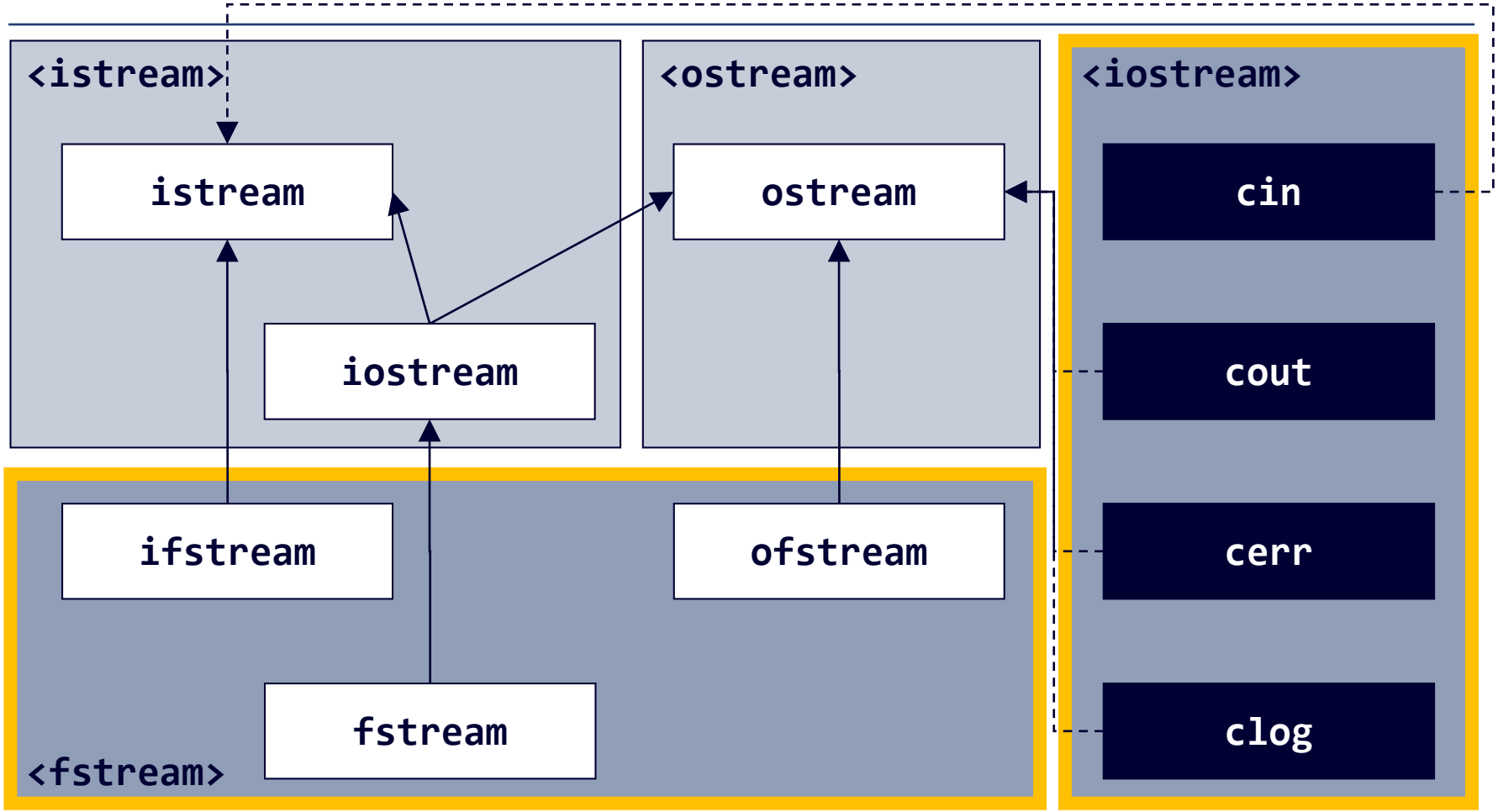


Fuente: George Hodan. «Computer Keyboard». Public Domain Pictures.net. <https://www.publicdomainpictures.net/en/view-image.php?image=164106> CC0 Public Domain,

Herramientas C++ para E/S

- Biblioteca **<istream>**
  - 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

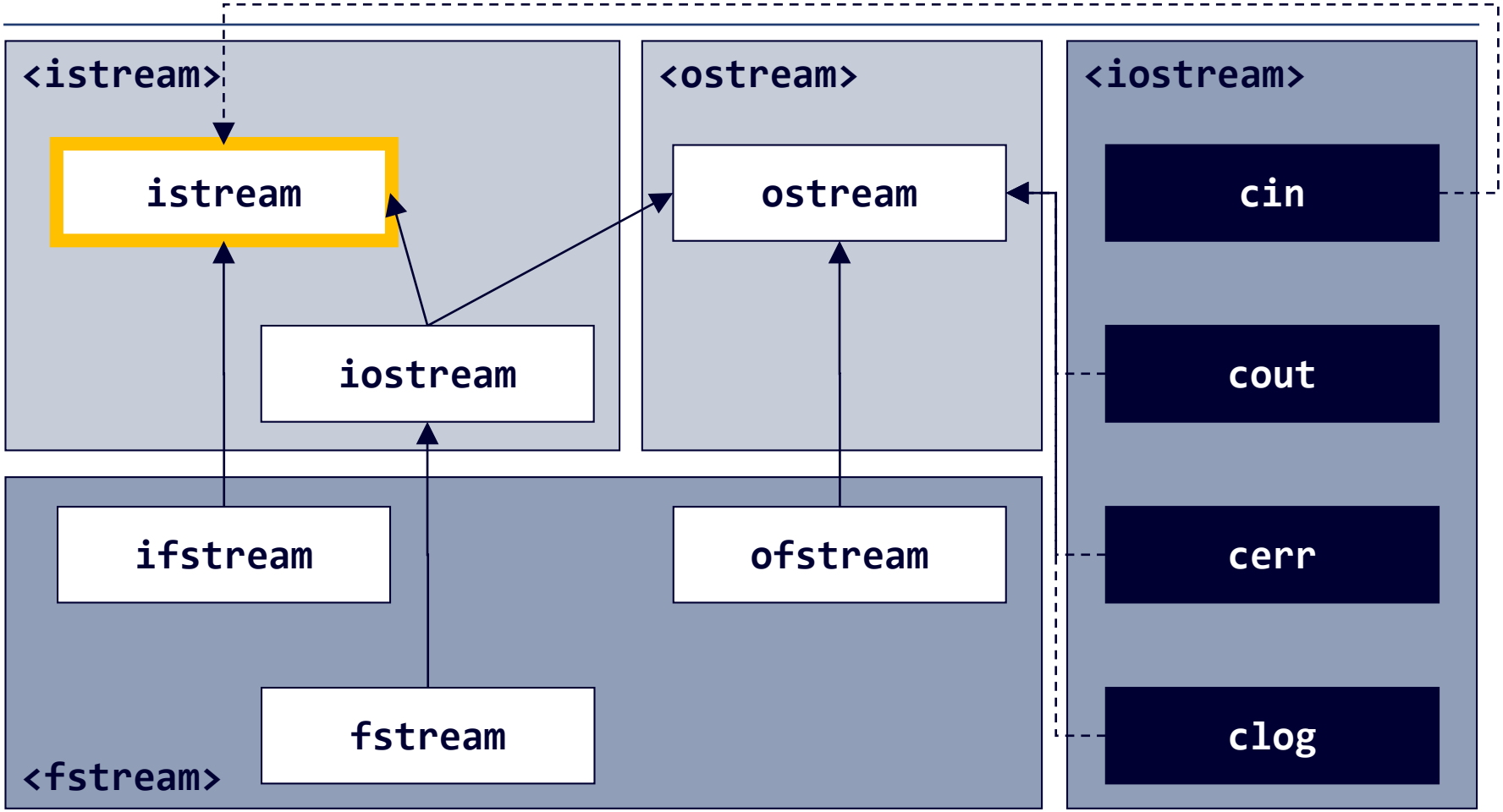
Herramientas C++ para E/S



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 de lectura
  - *Byte* a *byte*: `get`
  - Secuencia de *bytes* como *null terminated string* o como **`string`**: `getline`
- Operadores de extracción con formato (>>)
  - Mucho más potentes que los métodos básicos
  - Ya utilizados con `cin`

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`**
    - **`double`**
    - **`bool`**
    - **`string`**

Operador >> de extracción

Comportamiento general

- Se extraen del flujo e ignoran todos los caracteres blancos (espacios en blanco `' '`, tabuladores `'\t'` y finales de línea `'\n'`) que pueda haber antes del dato a extraer.
- Se extraen del flujo caracteres mientras pueden formar parte de un literal del tipo de que se trate.
- Se da valor a la variable de acuerdo con la representación formada por los caracteres extraídos.
- El primer carácter pendiente de leer del flujo es el que sigue al último carácter extraído del flujo y que se haya utilizado para dar valor a la variable.

Operador >> de extracción

Particularidades para enteros

- `cin >> variableEntera;`
- `f >> variableEntera;`

- Se extraen los blancos que pueda haber.
- Se extraen del flujo caracteres mientras pueden formar parte, en el orden correcto, de la representación de un entero válido (signo `'+'` o `'-'` y dígitos).
- Se da valor a `variableEntera` de acuerdo con el entero representado por los caracteres extraídos del flujo.
- El primer carácter pendiente de leer del flujo es el que sigue al último dígito extraído del flujo.

Operador >> de extracción
Particularidades para reales

- `cin >> variableReal;`
- `f >> variableReal;`

- Se extraen los blancos que pueda haber.
- Se extraen del flujo caracteres mientras pueden formar parte, en el orden correcto, de la representación de un real válido (signo, dígitos, punto decimal, exponente, …).
- Se da valor a `variableReal` de acuerdo con el número real representado por los caracteres extraídos del flujo.
- El primer carácter pendiente de leer del flujo es el que sigue al último dígito (o punto decimal) extraído del flujo y que formaba parte del real leído.

Operador >> de extracción

Particularidades para caracteres

- `cin >> variableCaracter;`
- `f >> variableCaracter;`

- Se extraen los blancos que pueda haber.
- Se extraen del flujo el primer carácter no blanco, 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

Particularidades para cadenas

- `cin >> variableCadena;`
- `f >> variableCadena;`

- Se extraen los blancos que pueda haber.
- Se extraen del flujo caracteres mientras sean distintos a espacio en blanco, tabulador o fin de línea.
- Se da valor a `variableCadena` de acuerdo con los caracteres no blancos extraídos del flujo.
- 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 caracter;
    string palabra;
    cin >> entero >> real >> caracter >> palabra;
    cout << "Los datos leídos son:" << endl;
    cout << "Entero: " << entero << endl;
    cout << "Real: " << real << endl;
    cout << "Carácter: '" << caracter << '\'' << endl;
    cout << "Palabra: \"" << palabra << '\"' << endl;
    return 0;
}
```

Métodos básicos de la clase `istream`

- Dado un objeto `f` de la clase **`istream`**
  - `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 del teclado se ha escrito:
  `Prog 1`
- Las invocaciones a `cin.get(c)` del siguiente código producirían los siguientes resultados:

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

Ejemplo

- Si en el flujo de entrada f se encontrara la secuencia:
  `Prog 1`
- Las invocaciones a `f.get(c)` del siguiente código producirían los siguientes resultados :

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

Otro ejemplo

```
/*
 * Lee caracteres de la entrada estándar hasta
 * que reconoce el carácter terminador FIN.
 * Informa al usuario de los caracteres que va
 * leyendo y del número total de caracteres
 * leídos hasta llegar a encontrar el carácter
 * FIN.
 */
int main();
```

Otro ejemplo

```
const char FIN = '.';

int main() {
    unsigned numVeces = 0;
    char c;
    cout << "No voy a parar hasta que introduzcas '" << FIN << "': ";

    do{
        cin.get(c);
        cout << "Carácter leído: '" << c << "'" << endl;
        numVeces++;
    } while (c != FIN);

    cout << "Número de caracteres leídos hasta '" << FIN << "': ";
    cout << numVeces;

    return 0;
}
```

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`.

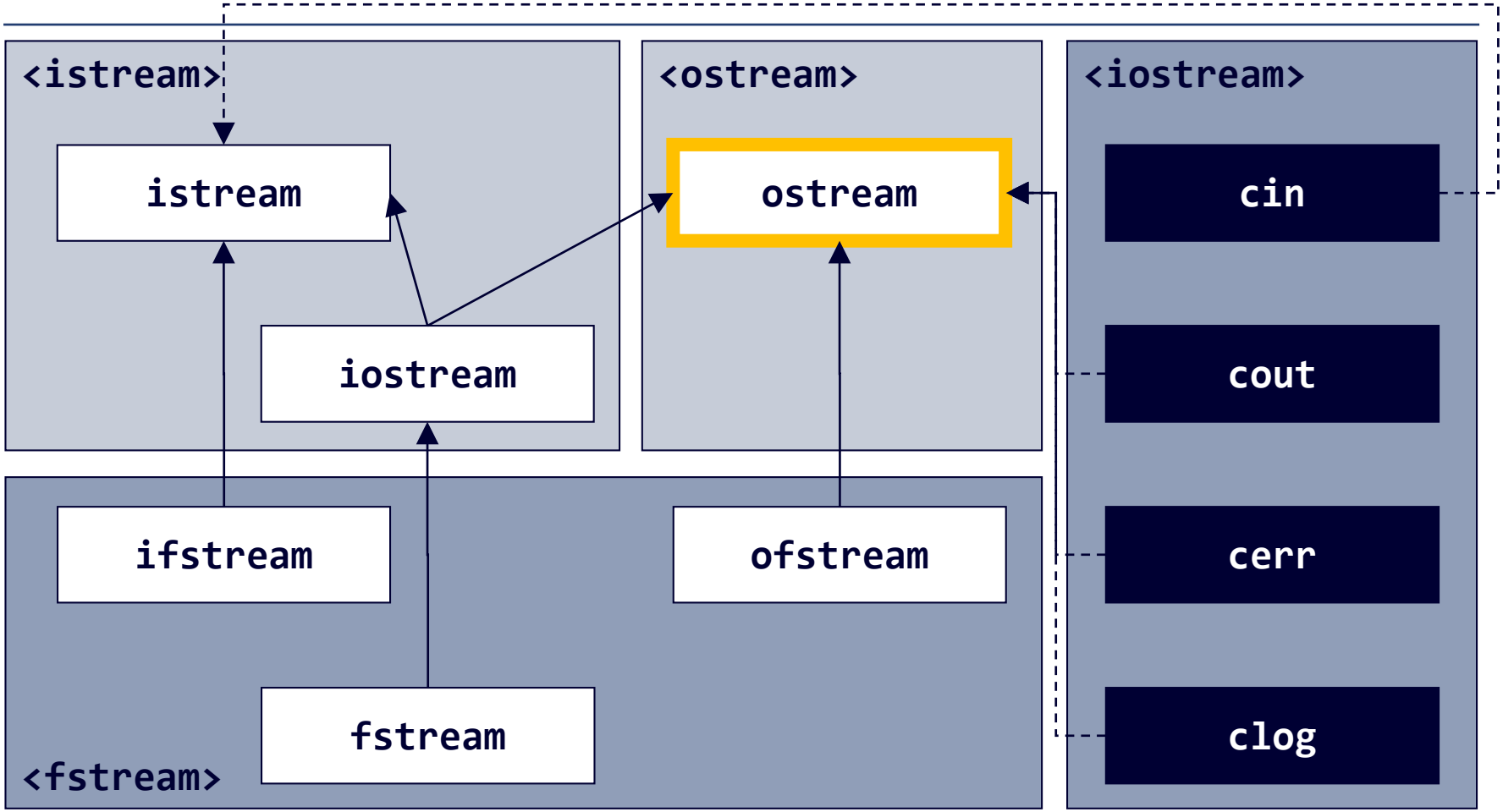
Ejemplo.
Función `getline`

```
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, '@');
    cout << "La línea escrita hasta la "
         << "primera '@' era \""
         << trozoDeLinea << "\"." << endl;
    getline(cin, trozoDeLinea);
    cout << "El resto de la línea era \""
         << trozoDeLinea << "\"." << endl;
    return 0;
}
```

Herramientas C++ para E/S



Clase `ostream`

- Definida en la biblioteca `<ostream>`
- Métodos básicos de escritura
  - *Byte* a *byte*: `put`
  - Secuencia de *bytes* como **`char`**`[]`: `write`
- Operadores de inserción con formato (`<<`)
  - Mucho más potentes que los métodos básicos
  - Ya utilizados con `cout`

Operaciones de la clase `ostream` para escritura con formato

- Operador de inserción **<<** para la escritura de una secuencia de datos a través de un flujo de salida
  - Ejemplo: `cout << d1 << d2 << d3;`
  - Disponible para:
    - **`char`**
    - **`int`** / **`unsigned`**
    - **`double`**
    - **`bool`**
    - **`string`**

Métodos básico de escritura de la clase `ostream`

- Método básico,
  dada la declaración **`ostream f;`**
  - `f.put(`**`char c`**`)`
    - Inserta el carácter `c`
      en el flujo de salida `f`

ostream
Ejemplo de put vs. <<

```
int main() {
    char caracter = 'E';

    cout << "Escritura de caracteres con put: ";
    cout.put('b');
    cout.put(caracter);
    cout << endl;

    cout << "Escritura de caracteres con <<: ";
    cout << 'b';
    cout << caracter;
    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 intermedio 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.
- Manipuladores de la biblioteca `<iomanip>`

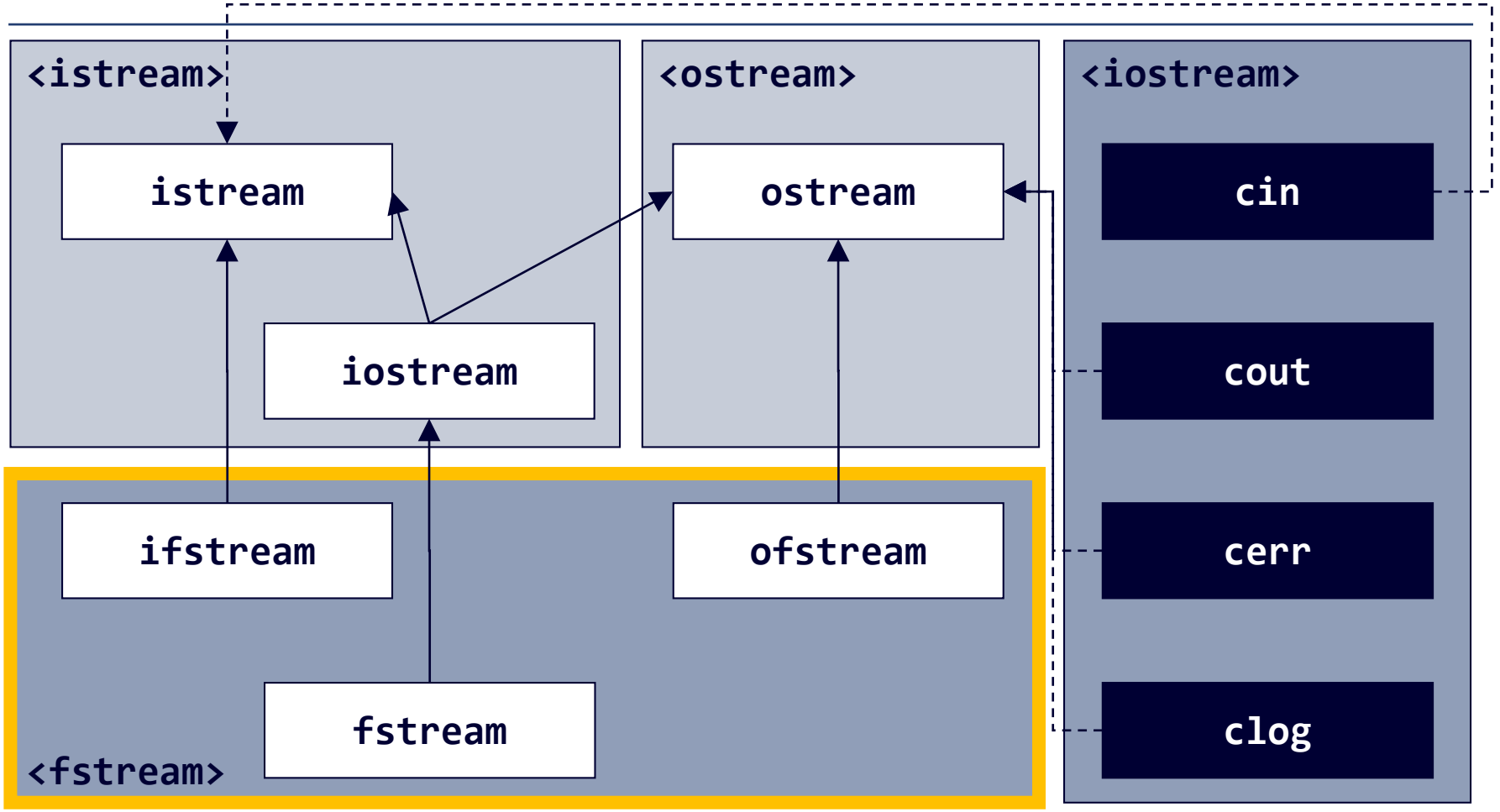
Resumen de las clases `istream` y `ostream`

- Operaciones para leer de un objeto `f` de la clase **`istream`**
  - `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

Funciones de la biblioteca `<fstream>`

- 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 alguna operación de lectura anterior no pudo completarse por no haber ya datos pendientes de lectura en el flujo `f`
  - Conversión implícita o explícita de `f` a **`bool`**
    - Se convierte en **`true`** si todas las operaciones anteriores con el flujo `f` se han realizado sin errores y **`false`** en caso contrario.

Esquema de trabajo con ficheros.
Escritura de datos en un fichero

- Declaración de un flujo con el que trabajar
- Asociar el flujo con el fichero externo
- Comprobar que la asociación ha sido correcta
- Mientras hay datos que escribir en el fichero
  - Escribirlos en el fichero a través del flujo
- Disociar el flujo del fichero externo

Ejemplo
Escritura de datos en un fichero

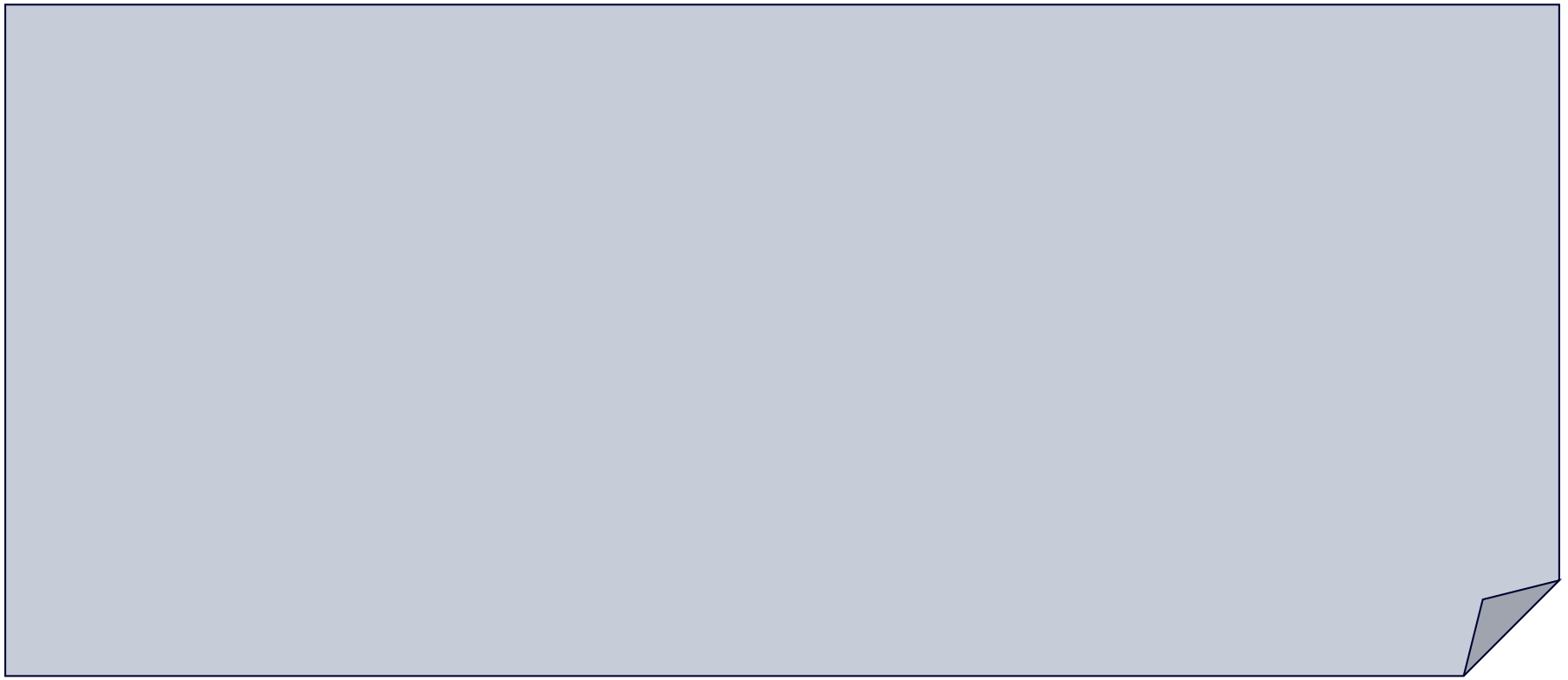
```
/*
 * Pre:  ---
 * Post: Crea un fichero denominado
 *       "mi-primer-fichero.txt" y escribe en
 *       él los números del 1 al 5, a razón de
 *       uno por línea. En caso de que se
 *       produzca un error, informa de
 *       ello escribiendo en «cerr».
 */
void crearFichero();
```

Ejemplo
Escritura de datos en un fichero

```
void crearFichero() {
    ofstream f;
    f.open("mi-primer-fichero.txt");
    if (f.is_open()) {
        for (unsigned i = 1; i <= 5; i++) {
            f << i << endl;
        }
        f.close();
    } else {
        cerr << "No se ha podido crear el fichero "
             << "\"miPrimerFichero.txt\"" << endl;
    }
}
```

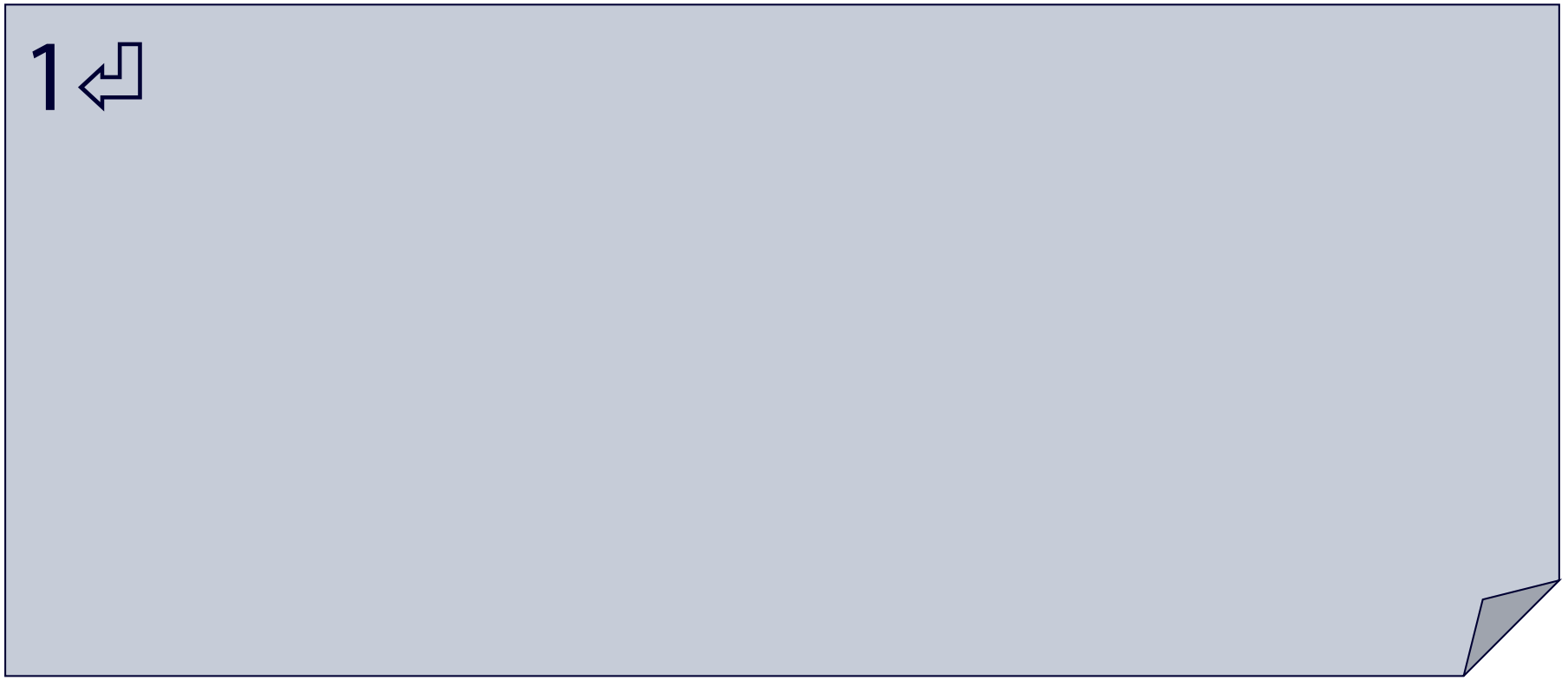
Ejemplo
Escritura de datos en un fichero

mi-primer-fichero.txt



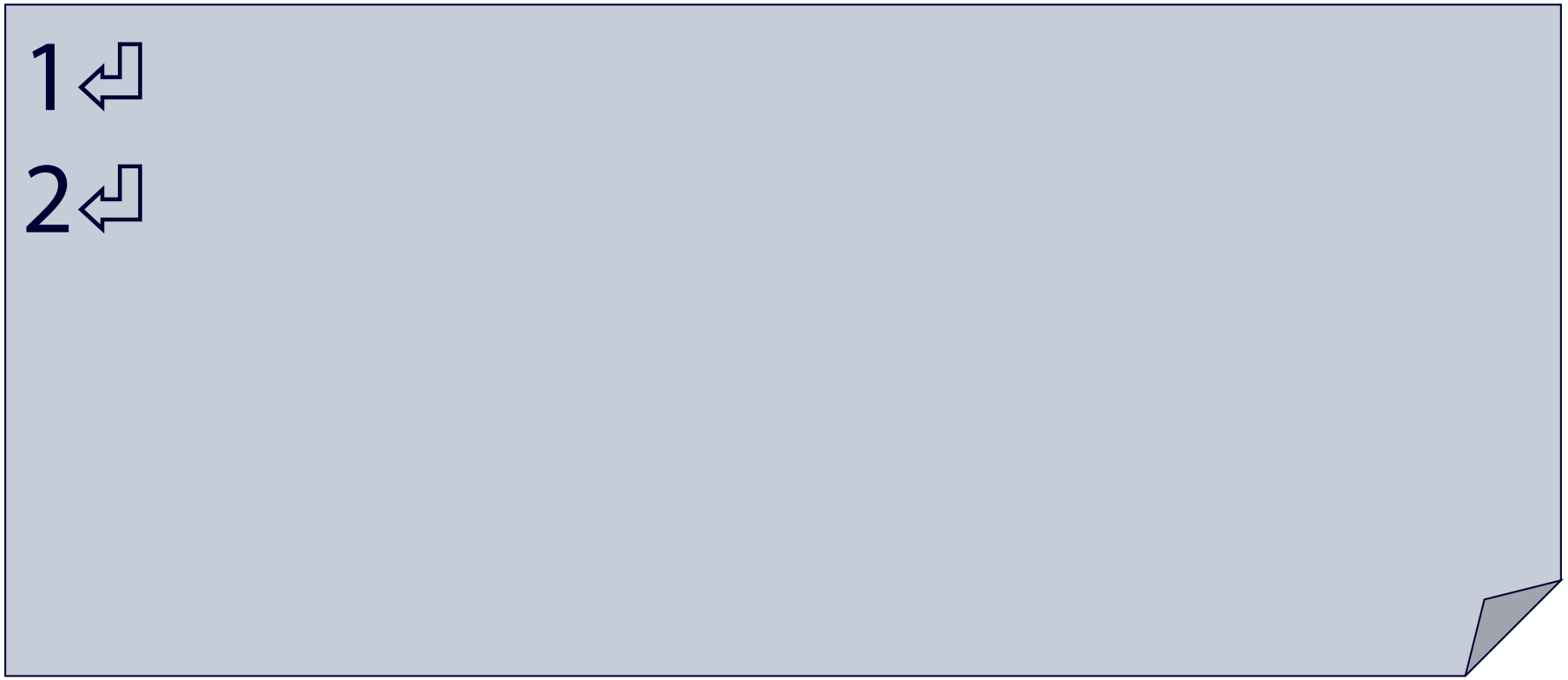
Ejemplo
Escritura de datos en un fichero

mi-primer-fichero.txt



Ejemplo
Escritura de datos en un fichero

mi-primer-fichero.txt



Ejemplo
Escritura de datos en un fichero

mi-primer-fichero.txt

1⏎
2⏎
3⏎

Ejemplo
Escritura de datos en un fichero

mi-primer-fichero.txt

1⏎
2⏎
3⏎
4⏎

Ejemplo
Escritura de datos en un fichero

mi-primer-fichero.txt

1⏎
2⏎
3⏎
4⏎
5⏎

Esquema de trabajo con ficheros.
Lectura de datos en un fichero

- Declarar un flujo con el que trabajar
- Asociar el flujo con el fichero externo
- Comprobar que la asociación ha sido correcta
- Mientras hay datos pendientes de leer en el fichero:
  - Leer un dato (o conjunto de datos)
  - Procesar el dato (o conjunto de datos)
- Disociar el flujo del fichero externo

Ejemplo
Lectura de datos de un fichero

```
/*
 * Pre:  «nombreFichero» es un fichero que contiene una
 *       secuencia de números enteros, a razón de uno por
 *       línea.
 * Post: Si «nombreFichero» define el nombre de un
 *       fichero, entonces muestra su contenido por
 *       pantalla, escribiendo cada entero del fichero
 *       separándolos entre sí con un espacio en blanco;
 *       en caso contrario advierte del error escribiendo
 *       un mensaje en la pantalla.
 */
void mostrar(const string nombreFichero);
```

Ejemplo
Lectura de datos de un fichero

```
void mostrar(const string nombreFichero) {
    ifstream f;
    f.open(nombreFichero);
    if (f.is_open()) {
        int n;
        while (f >> n) {
            cout << n << " ";
        }
        f.close();
    } else {
        cerr << "No se ha podido acceder a \""
             << nombreFichero << "\"" << endl;
    }
}
```

Ejemplo
Lectura de datos de un fichero

```
while (f >> n) {
    ...
}
```

- Se intenta leer de f para dar valor a n
- Si la lectura fue correcta
  - n = valor entero de lo leído
- f >> n, como expresión, se evalúa como el propio flujo f
- Como el propio flujo f se ha convertido en la condición de iteración, se convierte implícitamente a **bool**
- Si la lectura fue correcta
  - f se evalúa como **true**
- Si la lectura no fue correcta
  - f se evalúa como **false**

Ejemplo
Lectura de datos de un fichero

```
int main() {
    mostrar("mi-primer-fichero.txt");
    return 0;
}
```

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

bool(f): true

```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

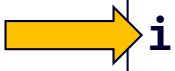
Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: ?

bool(f): true



```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: ?

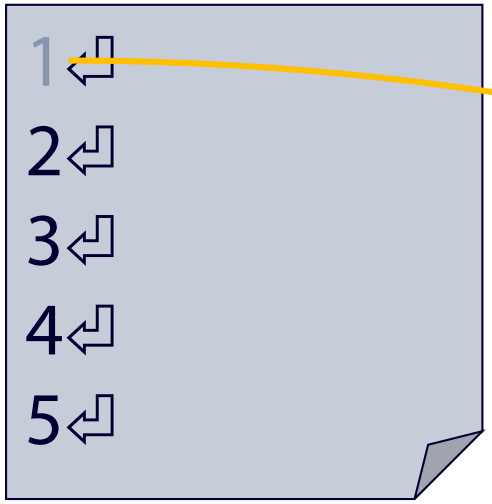
bool(f): true

```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:



Variable n: 1

bool(f): true

```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: 1

bool(f): true

```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: 1

bool(f): true

```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: 1

bool(f): true

```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

1

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: 1

bool(f): true

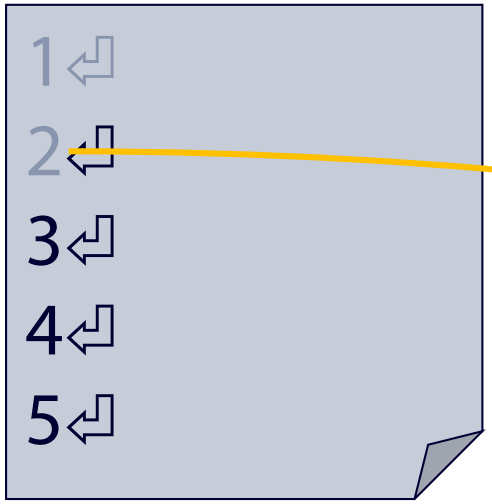
```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

1

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:



Variable n: 2

bool(f): true

```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

```
1
```

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: 2

bool(f): true

```
...
int n;
while (f >> n) {
    cout << n << " ";
}
...
```

Pantalla:

1

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: 2

bool(f): true

```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

1

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: 2

bool(f): true

```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

1 2

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: 2

bool(f): true

```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

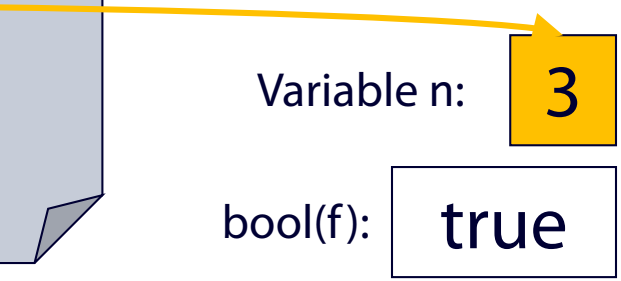
Pantalla:

1 2

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎




```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

1 2

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: 3

bool(f): true

```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

1 2

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: 3

bool(f): true

```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

1 2

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: 3

bool(f): true

```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

1 2 3

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: 3

bool(f): true

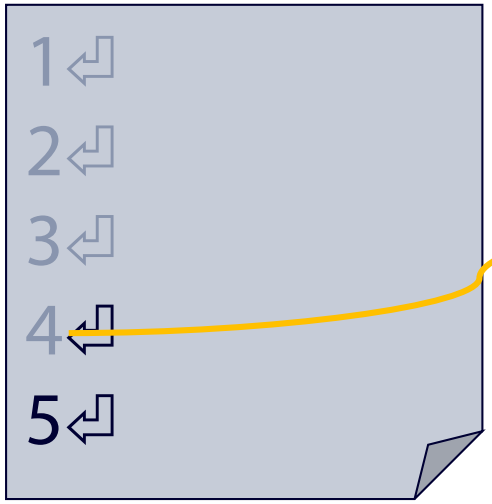
```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

1 2 3

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:



Variable n: 4

bool(f): true

```
...
int n;
while (f >> n) {
    cout << n << " ";
}
...
```

Pantalla:

```
1 2 3
```

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: 4

bool(f): true

```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

1 2 3

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: 4

bool(f): true

```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

1 2 3

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: 4

bool(f): true

```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

1 2 3 4

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: 4

bool(f): true

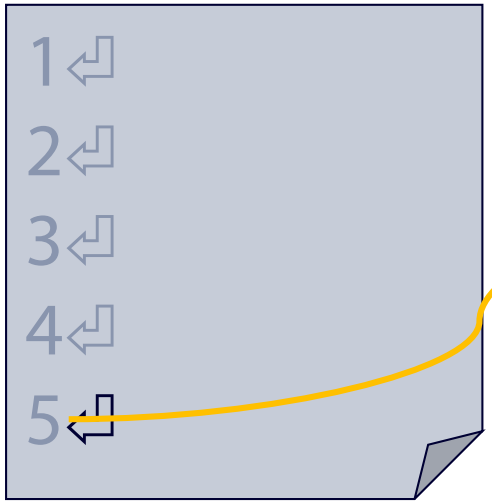
```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

1 2 3 4

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:



Variable n: 5

bool(f): true

```
...
int n;
while (f >> n) {
    cout << n << " ";
}
...
```

Pantalla:

```
1 2 3 4
```

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: 5

bool(f): true

```
...
int n;
while (f >> n) {
    cout << n << " ";
}
...
```

Pantalla:

1 2 3 4

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: 5

bool(f): true

```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

1 2 3 4 5

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: 5

bool(f): true

```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

1 2 3 4 5

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: ?

bool(f): true

```
...
int n;
while (f >> n) {
    cout << n << " ";
}
...
```

Pantalla:

1 2 3 4 5

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: ?

bool(f): false

```
...
int n;
while (f >> n) {
    cout << n << " ";
}
...
```

Pantalla:

1 2 3 4 5

Ejemplo
Lectura de datos de un fichero

mi-primer-fichero.txt:

1⏎
2⏎
3⏎
4⏎
5⏎

Variable n: ?

bool(f): false

```
...
int n;
while (f >> n) {
   cout << n << " ";
}
...
```

Pantalla:

1 2 3 4 5

Ejemplo. Copia

```
/*
 * Pre:  «nombreFichero» es un fichero que contiene
 *       una secuencia de números enteros, a razón
 *       de uno por línea.
 * 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 original;
    original.open(nombreFichero);
    if (original.is_open()) {
        ofstream copia;
        copia.open(nombreCopia);
        if (copia.is_open()) {
            int n;
            while (original >> n) {
                copia << n << endl;
            }
            copia.close();
        } else {
            cerr << "No se ha podido escribir en \"" << nombreCopia
                 << "\"." << endl;
        }
        original.close();
    } else {
        cerr << "No se ha podido acceder a \"" << nombreFichero << "\"."
             << endl;
    }
}
```

Ejemplo. Copia

```
int n;
while (original >> n) {
    copia << n << endl;
}
```

Resumen bibliotecas <istream>, <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:	Por ser un ostream:
f.get(**char** &c) getline(**istream** &f, **string** &cad) getline(**istream** &f, **string** &cad, **char** delimitador) f >> *variable_char* f >> *variable_int* f >> *variable_double* f >> *variable_string*	g.put(**char** c) g << *expresión_char* g << *expresión_int* g << *expresión_double* g << *expresión_cadena* g << flush g << endl
Por ser un ifstream:	Por ser un ofstream:
f.open(**string** nombreFichero) f.is_open() f.close() **bool** f.eof() Conversión de f a **bool**	g.open(**string** nombreFichero) g.is_open() g.close()

¿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.
- Clases de problemas
- Práctica 6
- Trabajos y exámenes de cursos anteriores.