

# Programación 1

## Tema 13

---

### Entrada y salida de datos



Escuela de  
Ingeniería y Arquitectura  
**Universidad** Zaragoza



# Objetivos

---

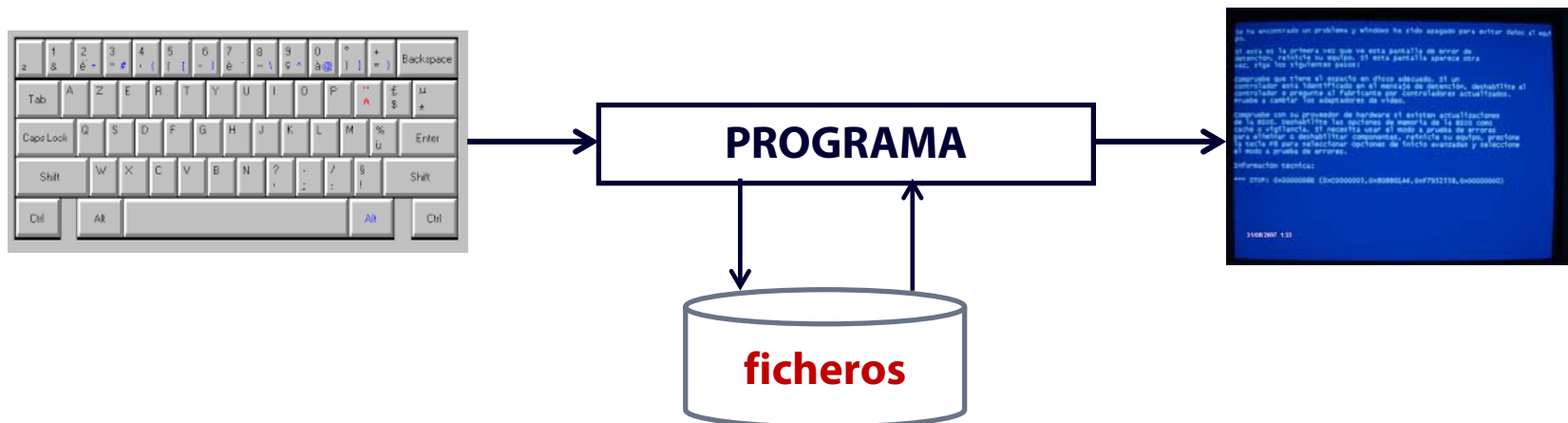
- ❑ Interacción de un programa con su entorno (consola, sistema de ficheros) leyendo o escribiendo datos
- ❑ Fichero como secuencia persistente de datos
- ❑ Herramientas de C++ para entrada y salida de datos

# 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

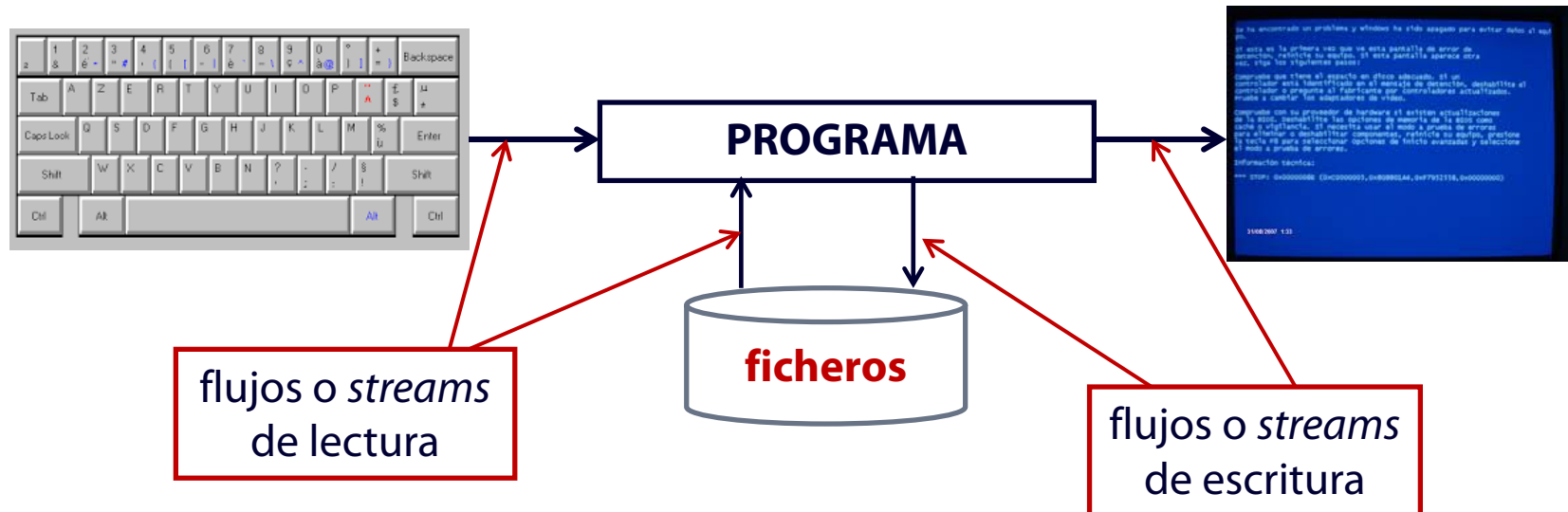


# Flujos

---

- 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



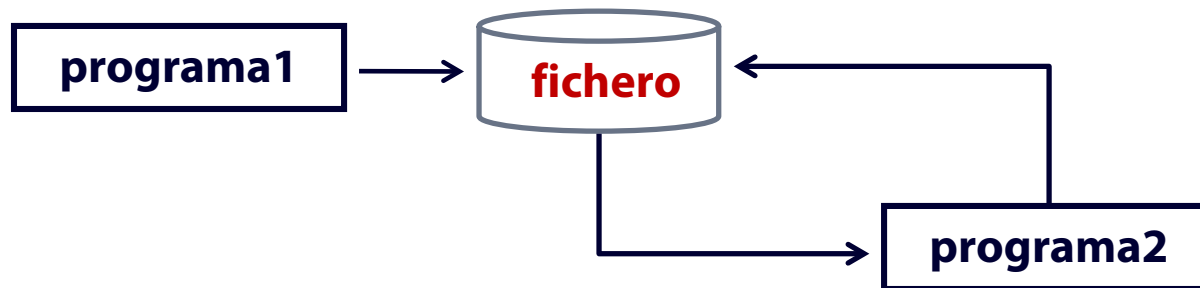
# Ficheros o archivos de datos

---

- 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.



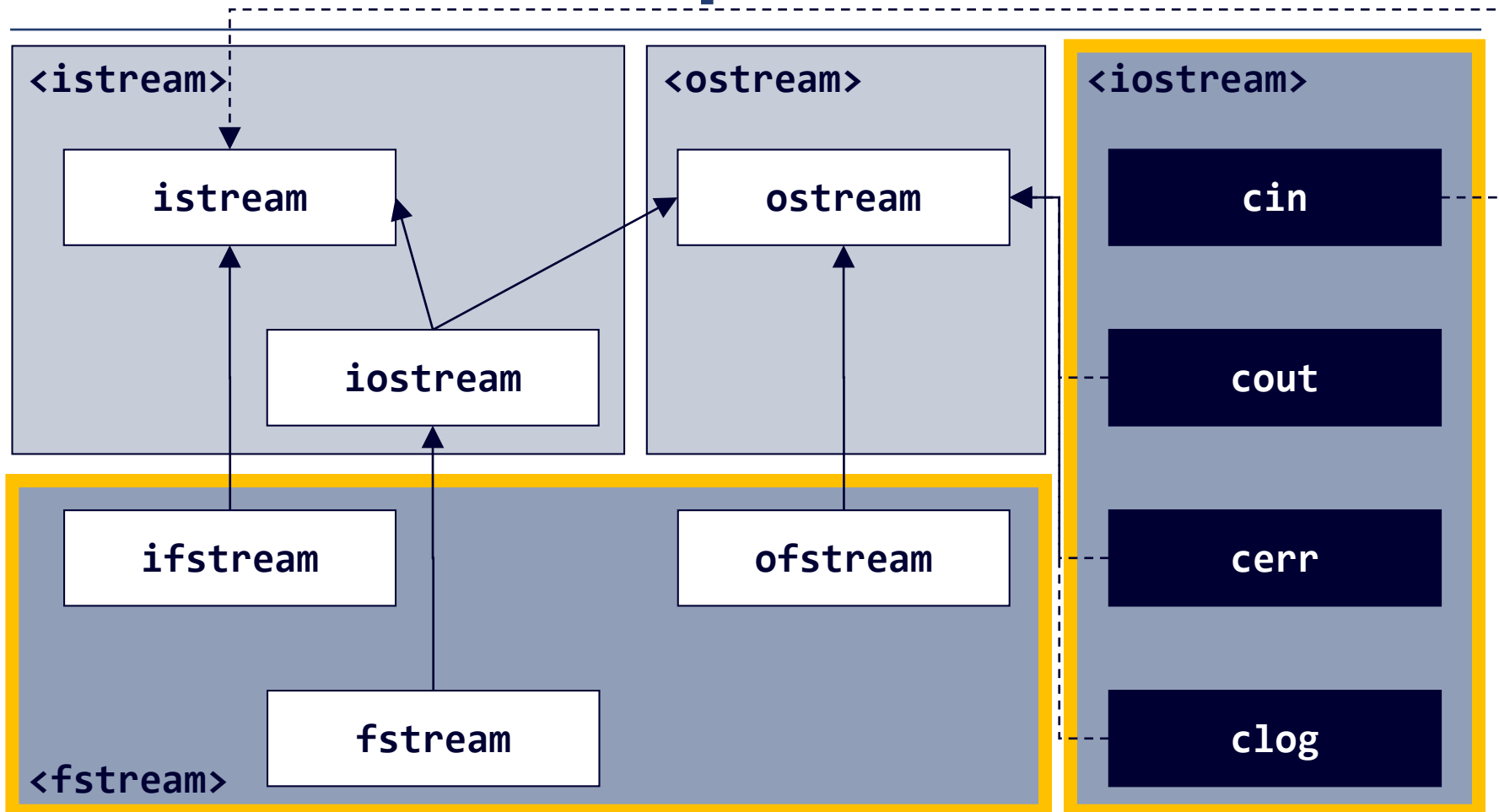


# Herramientas C++ para E/S

---

- Biblioteca **<istream>**
  - Flujos de entrada de la clase **istream** para la lectura de datos
  - Flujos de entrada y salida de la clase **iostream** para la lectura y la escritura 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 datos de un fichero
  - Flujos de salida de la clase **ofstream** para la escritura de datos en un fichero
  - Flujos de entrada y salida de la clase **fstream** para la lectura de datos de un fichero y la escritura en el mismo fichero

# 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)

# Biblioteca <iostream>

---

- Define la clase `istream`
- Operador de extracción `>>` para la lectura de una secuencia de datos a través de un flujo de entrada:
  - Ejemplo: `cin >> v1 >> v2 >> v3;`
- Otras funciones, dada la declaración `istream f;`
  - **`f.get()`**
    - Extrae el siguiente carácter pendiente de leer del flujo de entrada `f` y lo devuelve como resultado
  - **`f.get(char& c)`**
    - Extrae el siguiente carácter pendiente de leer del flujo en entrada `f` y lo asigna al parámetro `c`

## Biblioteca <iostream>

---

- **f.get(char sec[], streamsize n)**
  - Extrae una secuencia de hasta  $n - 1$  caracteres del flujo de entrada *f* y la asigna al vector *sec*. Extrae, como mucho,  $n - 1$  caracteres y, como mucho, hasta que se encuentra con el carácter '`\n`', que **no** extrae.
- **f.get(char sec[], streamsize n, char delimitador)**
  - Extrae una secuencia de caracteres del flujo de entrada *f* y la asigna al vector *sec*. Extrae, como mucho,  $n - 1$  caracteres, hasta que se encuentra con el carácter *delimitador*, que **no** extrae.

# Biblioteca <iostream>

---

- **fgetline(char sec[], streamsize n)**
  - Extrae una secuencia de caracteres del flujo de entrada f y la asigna al vector sec. Extrae, como mucho, n – 1 caracteres y, como mucho, hasta que se encuentra con el carácter '\n', que **sí** extrae.
- **fgetline(char sec[], streamsize n, char delimitador)**
  - Extrae una secuencia de caracteres del flujo en entrada f y la asigna al vector sec. Extrae, como mucho, n – 1 caracteres, hasta que se encuentra con el carácter delimitador, que **sí** extrae.

# Biblioteca <ostream>

---

- Define la clase `ostream`
- Operador de inserción `<<` para la escritura de una secuencia de datos a través de un flujo de salida:
  - Ejemplo: `cout << d1 << d2 << d3;`
- Otras funciones, dada la declaración `ostream f;`
  - **`f.put(char c)`**
    - Inserta el carácter `c` en el flujo de salida `f`
  - **`f.write(const char sec[],  
                                streamsize n)`**
    - Inserta los `n` primeros caracteres del vector `sec` en el flujo en salida `f`

# Biblioteca <ostream>

---

- Manipuladores

- **flush**

- vacía el buffer asociado al flujo de salida

- **endl**

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

- **ends**

- Inserta el carácter nulo '`\0`' en el flujo de salida



# Resumen bibliotecas

## <iostream> y <ostream>

- Operaciones para leer de un objeto `f` de la clase `istream`
  - `int f.get()`
  - `f.get(char& c)`
  - `f.get(char sec[], streamsize n)`
  - `f.get(char sec[], streamsize n, char delimitador)`
  - `f.getline(char sec[], streamsize n)`
  - `f.getline(char sec[], streamsize n, char delimitador)`
  - `f >> variable_char`
  - `f >> variable_int`
  - `f >> variable_double`
  - `f >> variable_char[]`

# Resumen bibliotecas

## <iostream> y <ostream>

---

- Operaciones para escribir en un objeto f de la clase ostream
  - f.put(char c)
  - f.write(const char sec[], streamsize n)
  - f << *expresión\_char*
  - f << *expresión\_int*
  - f << *expresión\_double*
  - f << *expresión\_cadena*
  - f << flush
  - f << endl
  - f << ends

# 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 char 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("miPrimerFichero.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

## 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 char nombreFichero[]);
```



# Ejemplo

## Lectura de datos de un fichero

```
void mostrar (const char 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

```
void mostrar (const char 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 char nombreFichero[],  
            const char nombreCopia[]);
```

## Ejemplo. Copia

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

## Ejemplo. Copia (línea a línea)

```
const int MAX_LONG_LINEA = 1024;

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