Metodología de la Programación II

Gestión de E/S.

Ficheros

Objetivos

1. Conocer el concepto de flujo como el mecanismo abstracto de comunicación entre la memoria y los dispositivos.

2. Conocer cómo se realiza la E/S con *buffer*.

3. Conocer cómo personalizar la manera de realizar E/S.

4. Entender que la E/S con ficheros no es más que una *especialización* de la E/S con los dispositivos estándar (teclado y consola).

5. Saber cómo realizar E/S sin formato.

5.1 Flujos

- La E/S se produce en forma de **flujos**.
- **Flujo**: Una secuencia de bytes.
  - Operación de entrada, los bytes *fluyen* de un dispositivo hacia la memoria.
  - Operación de salida, los bytes *fluyen* de la memoria hacia un dispositivo.
- La aplicación es quien da significado al contenido (bytes) del flujo.
- Un flujo es un *objeto* que permite la comunicación entre:
  1. Memoria y dispositivos de E/S.
  2. Memoria y ficheros.

- **Flujos por defecto**.

  Al empezar la ejecución de un programa en C++ se dispone de tres flujos:

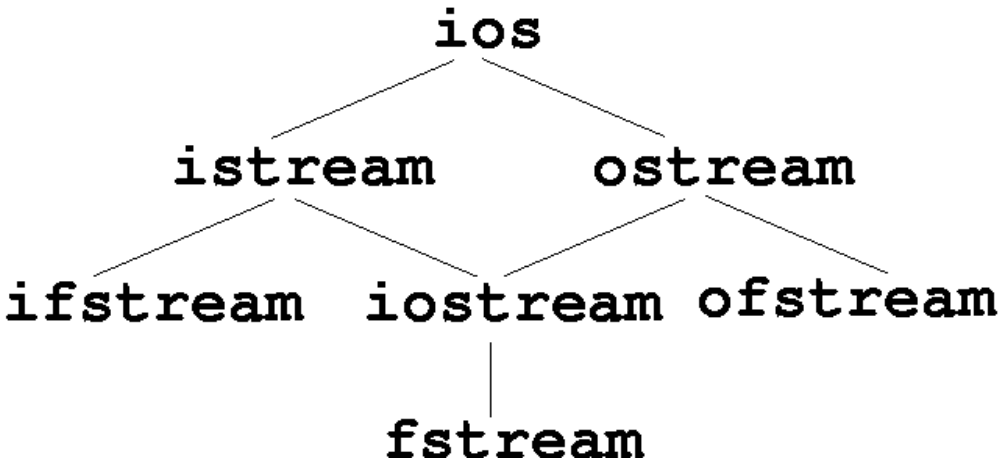
  - `cin`. Entrada estándar (*teclado*).
  - `cout`. Salida estándar (*consola*).
  - `cerr`. Salida de errores estándar (*consola*).

  siempre que se incluya `iostream`

- **Ficheros de cabecera para E/S**.

  - `iostream`. Operaciones básicas de E/S y declaración de los objetos `cin`, `cout` y `cerr`.
  - `iomanip`. Para E/S con formato (personalizar la E/S).
  - `fstream`. Para E/S con ficheros.

- **Clases y objetos de E/S de flujo**.



Clases:

- `ostream`: Operaciones de salida.
  `cout` y `cerr` son objetos de la clase `ostream`.
- `istream`. Operaciones de entrada.
  `cin` es un objeto de la clase `istream`.
- `iostream`. Operaciones de E/S.
- `ifstream`. Operaciones de E desde un fichero.
- `ofstream`. Operaciones de S hacia un fichero.
- `fstream`. Operaciones de E/S con ficheros.

5.2 Flujos de Salida

- Clase `ostream`.

- Facilidades para:
  - Salida de datos de tipos estándar con el operador (**sobrecargado**) <<
  - Salida de caracteres con `put()`
  - Salida sin formato con `write()`.
  - Salida de enteros en dec., octal y hex.
  - Salida de números reales con diversas precisiones, con puntos decimales o en notación científica.
  - Salidas con campos de anchuras prefijadas y posibilidad de rellenar los "huecos" con caracteres.

. . . . . .

5.2.1 El operador <<

- Se evalúa de izquierda a derecha.
- Puede recibir como argumento alguno de los tipos para los que está sobrecargado, devuelve una *referencia* a un objeto de la clase `ostream`, que convierte los valores de los tipos recibidos en caracteres y los envía a la salida.
- Está **sobrecargado** para:
  - Tipos estándar (tipos predefinidos de C++: `int`, `float`, . . .).
  - Cadenas (literales) de caracteres.
  - Punteros.

Ejemplo que demuestra la sobrecarga de <<

```
// Fichero: demo_cout.cpp
#include <iostream>
using namespace std;
int main ()
{
  int n = 5;
  float r = 7.33;
  char  *cad = "Hola";
  int   *pi = &n;
  float *pf = &r;

  cout << "Literal de cadena" << endl;
  cout << n << endl;
  cout << r << endl;
  cout << *pi << "  " << pi << endl;
```

```
  cout << *pf << "  " << pf << endl;
  cout << cad << "  " << static_cast<void *>(cad)<<endl;
  return (0);
}

% demo_cout
Literal de cadena
5
7.33
5  0xbffff834
7.33  0xbffff830
Hola  0x8048908
```

5.2.2 Función `put()`

```
ostream & put (char c);
```

La función `put()` recibe un argumento de tipo carácter y lo pone en el flujo de salida sobre el que se aplica. Devuelve una referencia a `ostream`.

```
// Fichero: demo_put.cpp
#include <iostream>
using namespace std;
int main ()
{
   char c1='H', c2='o', c3='l', c4='a';
   cout << c1 << c2 << c3 << c4 << endl;
   cout.put (c1);
```

```
   cout.put (c2);
   cout.put (c3);
   cout.put (c4);
   cout.put ('\n');
   cout.put(c1).put(c2).put(c3).put(c4).put('\n');
   return (0);
}

% demo_put
Hola
Hola
Hola
```

5.3 Flujos de Entrada

- Clase `istream`.
- Facilidades para:
  - Lectura de datos de tipos estándar con el operador (**sobrecargado**) >>
  - Entrada de caracteres con `get()`
  - Entrada de líneas con `getline()`
  - Entrada sin formato con `read()`.
  - Consultar el *fin de fichero* con `eof()`.

. . . . . .

5.3.1 El operador >>

- Se evalúa de izquierda a derecha.
- Forma "palabras" extrayendo caracteres del flujo: considera que los espacios son *separadores*.
- Si encuentra el carácter *fin de fichero* (`EOF`) devuelve `0`. Si no, convierte la "palabra" al tipo adecuado y lo almacena en memoria, devolviendo una referencia a un objeto de la clase `istream`.

```
// Fichero:  demo_cin.cpp
#include <iostream>
using namespace std;

int main ()
{
   int n1, n2;

   cout << "\nIntroducir los valores de dos enteros: ";
   cin >> n1 >> n2;
   cout << "   La suma es: " << (n1 + n2) << endl;
   cout << "   " << n1 << ((n1 == n2)? " es" : " no es");
   cout << " igual a " << n2 << endl;

   return (0);
}
```

Ejemplo que demuestra la sobrecarga de <<

```
// Fichero:  demo_cin2.cpp
#include <iostream>
using namespace std;

int main ()
{  int n;
   float r;
   char cad[50];
   cout << "\nIntroducir valores para: \n";
   cout << "   Entero: ";
   cin  >> n;
   cout  << "   Real: ";
   cin  >> r;
   cout  << "   Cadena:  ";
   cin >> cad;
```

```
   cout << "\nValores leidos: \n";
   cout << "   Entero: " << n << endl;
   cout  << "   Real: " << r << endl;
   cout  << "   Cadena:  " << cad << endl << endl;
   return (0);
}

% demo_cin2
Introducir valores para:
   Entero: 5
   Real: 7.33
   Cadena:  Una cadena

Valores leidos:
   Entero: 5
   Real: 7.33
   Cadena:  Una
```

5.3.2 La función `eof()`

```
bool eof ()
```

Es una función miembro de la clase `ios`.

Devuelve `true` si en el flujo se encuentra el fin de fichero.

El carácter `EOF` puede introducirse en `cin` pulsando `Ctrl+D` (Linux) o `Ctrl+Z` (MS-DOS).

```
// Fichero:  suma_int.cpp
#include <iostream>
using namespace std;

int main ()
{ int n, cont=0, sum=0;
   cout << "\n(Suma acumulada = " << sum << "). ";
   cout << "Introducir entero n. " << cont+1 << " : ";
   cin >> n;

   while (!cin.eof()){
      sum += n;
      cont++;
      cout << "(Suma acumulada = " << sum << "). ";
      cout << "Introducir entero n. "<< cont+1 << " : ";
      cin >> n;
   }
```

```
   cout << "\n\nLa suma total de los " << cont;
   cout << " enteros introducidos es " << sum;
   cout << endl;
   return (0);
}
```

```
% suma_int
(Suma acumulada = 0). Introducir entero n. 1 : 1
(Suma acumulada = 1). Introducir entero n. 2 : 2
(Suma acumulada = 3). Introducir entero n. 3 : 3
(Suma acumulada = 6). Introducir entero n. 4 : (CTRL+D)

La suma total de los 3 enteros introducidos es 6
```

El operador >> devuelve 0 cuando encuentra el *fin de fichero* en el flujo de entrada sobre el que se aplica.

```
// Fichero:  suma_int2.cpp
#include <iostream>
using namespace std;

int main ()
{
   int n, cont=0, sum=0;

   cout << "\n(Suma acumulada = " << sum << "). ";
   cout << "Introducir entero n. " << cont+1 << " : ";
   while ((cin >> n) != 0) { // while (cin >> n)
      sum += n;
      cont++;
      cout << "(Suma acumulada = " << sum << "). ";
      cout << "Introducir entero n. " << cont+1 << " : ";
   }
```

```
   cout << "\n\nLa suma total de los " << cont;
   cout << " enteros introducidos es " << sum;
   cout << endl;

   return (0);
}

% suma_int2
(Suma acumulada = 0). Introducir entero n. 1 : 1
(Suma acumulada = 1). Introducir entero n. 2 : 2
(Suma acumulada = 3). Introducir entero n. 3 : 3
(Suma acumulada = 6). Introducir entero n. 4 : (CTRL+D)

La suma total de los 3 enteros introducidos es 6
```

5.3.3 Funciones `get()` y `getline()`

- **Función `get()` sin argumentos**.

  `int get ()`

  Lee un carácter desde el flujo, devolviendo el carácter encontrado. Si encuentra el carácter *fin de fichero* devuelve `EOF`.

```
// Fichero: eco.cpp
#include <iostream>
using namespace std;

int main ()
{
   char c;
   while ((c = cin.get()) != EOF)
      cout.put (c);
   return (0);
}

% eco
Esto es una prueba de entrada
Esto es una prueba de entrada
con la funcion get (CTRL+D)
con la funcion get
```

- **Función** `get()` **con un argumento**.

  `istream & get (char & c)`

  Lee un carácter desde el flujo y lo guarda en la variable que recibe como argumento. Devuelve `0` si encuentra el carácter *fin de fichero*. En otro caso, devuelve una referencia al objeto de la clase `istream` con el que se llamó.

```
// Fichero:  demo_get.cpp
#include <iostream>
using namespace std;
int main ()
{  char c1, c2, c3, c4, c5, c6;

   cout << "Introduce una palabra que tenga 6 o más caracteres: ";
   cin.get (c1);
```

```
    cin.get (c2);
    cin.get (c3);
    cin.get(c4).get(c5).get (c6);

    cout << "\nLos tres primeros: ";
    cout.put(c1).put(c2).put(c3).put('\n');
    cout << "Los tres siguientes: ";
    cout.put(c4).put(c5).put(c6).put('\n');
    return (0);
}
```

```
% demo_get
Introduce una palabra que tenga 6 o mas caracteres:
estereotipo

Los tres primeros: est
Los tres siguientes: ere
```

- **Lectura de cadenas: Función `get()` con dos/tres argumentos y función `getline()`.**

  - El operador >> está sobrecargado para poder leer una *palabra*.
  - `get()` con dos o tres parámetros:

```
istream & get (char *p,int n,char del='\n');
```

  Lee, como mucho, `n-1` caracteres desde el flujo de entrada. Termina antes si encuentra el carácter `del` (por defecto, `'\n'`).

  No inserta `del`. En su lugar, inserta el terminador de cadena `'\0'`

  $\Rightarrow$ `del` permanece en el flujo.

```
// Fichero:  demo_get_cadenas.cpp

#include <iostream>
using namespace std;
int main ()
{
   const int TAM = 80;
   char cad1[TAM], cad2[TAM];

   cout << endl;
   cout << "Introducir una cadena: ";
   cin.get (cad1, TAM);

   cout << "Introducir otra cadena: ";
   cin.get (cad2, TAM);

   cout << endl;
```

```
    cout << "\nPrimera cadena : " << cad1;
    cout << "\nSegunda  cadena : " << cad2;
    cout << endl;
    return (0);
}
```

```
% demo_get_cadenas
Introducir una cadena: esta es la primera
Introducir otra cadena:

Primera cadena : esta es la primera
Segunda  cadena :
```

- Función `ignore()`

  `istream & ignore (int n=1, char del=EOF);`
  Descarta, como mucho, `n` caracteres del flujo de entrada (por defecto, 1).
  Termina antes si encuentra `del`, que también lo descarta (por defecto, `EOF`).

```
   ................
cout << "Introducir una cadena: ";
cin.get (cad1, TAM);

cin.ignore(TAM, '\n');  // Elimina '\n' de cin

cout << "Introducir otra cadena: ";
   ................
```

```
// Fichero:  lee_y_suma.cpp
#include <iostream>
#include <cstdlib>
using namespace std;
int main ()
{
   const int TAM = 256;
   const int NUM_LIN = 3;
   char cad[TAM];
   int suma, cont;

   while (!cin.eof()) {
      for (cont=0,suma=0; cont<NUM_LIN; cont++) {
         cin.get (cad, TAM, ' ');
         cin.ignore (TAM, ' ');
         suma += atoi(cad);
      }
      if (!cin.eof()) {
         cout << "Suma: " << suma << endl;
```

```
         cin.ignore (TAM, '\n');
      } // if
   } // while
   return (0);
}
```

```
% lee_y_suma
1 2 3
Suma = 6
4 5 6
Suma = 15
(CTRL+D)
```

- Función `getline()`

```
istream & getline (char *p,int n,char del='\n');
```

Como `get()`, pero elimina el delimitador del flujo de entrada.

```
// Fichero:  demo_getline_cadenas.cpp

#include <iostream>
using namespace std;
int main ()
{
   const int TAM = 80;
   char cad1[TAM], cad2[TAM];

   cout << endl;
   cout << "Introducir una cadena: ";
```

```
    cin.getline (cad1, TAM);

    cout << "Introducir otra cadena: ";
    cin.getline (cad2, TAM);

    cout << endl;
    cout << "\nPrimera cadena : " << cad1;
    cout << "\nSegunda  cadena : " << cad2;
    cout << endl;

    return (0);
}
```

```
% demo_getline_cadenas

Introducir una cadena: esta es la primera
Introducir otra cadena: y esta, la segunda


Primera cadena : esta es la primera
Segunda  cadena : y esta, la segunda
```

Para leer cadenas se aconseja el uso de `getline()` frente a `get()`.

5.3.4 Funciones `peek()` y `putback()`

- `peek()` consulta el siguiente carácter del flujo de entrada y lo devuelve, pero **no** lo extrae de éste.

- `putback()` *devuelve* al flujo de entrada el carácter obtenido previamente con un `get()`.

5.4 Evaluación del estado de un flujo

Los objetos de `iostream` mantienen una serie de indicadores o *banderas* que informan del estado del flujo.

Se consultan a través de las funciones miembro (booleanas):

- `eof()` Devuelve `true` si el objeto `istream` sobre el que se aplica encuentra `EOF`.
- `bad()` Devuelve `true` si se intenta realizar una operación no válida.
- `fail()` Devuelve `true` si `bad()` es `true` o cuando falla una operación.
- `good()` Devuelve `true` si todas las anteriores son `false`

`int clear()` borra todos los indicadores de error que estuvieran activados.

5.5 E/S sin formato

Funciones `read()`, `write()` y `gcount()`.

```
istream & read (char *p, int n);
```

Lee del flujo de entrada `n` bytes y los copia en el *buffer* cuya dirección indica `p`.

```
int gcount ();
```

Devuelve el número de caracteres leidos en la última operación de entrada.

```
ostream & write (const char *p, int n);
```

Escribe en el flujo de salida `n` bytes, que toma del *buffer* cuya dirección indica `p`.

```
// Fichero: tamanio.cpp
#include <iostream>
using namespace std;
int main ()
{  const int TAM_BUFFER = 10;
   char buffer[TAM_BUFFER];
   int tam = 0;
   while (cin.read(buffer, TAM_BUFFER)) {
      tam += TAM_BUFFER;
   }
   tam += cin.gcount();
   cout << "\nTamanio = " << tam << endl;
   return (0);
}
```

```
% tamanio
Este.es.un.ejemplo(ENTER)
para.tamanio(CTRL+D)
Tamanio = 31
```

```
// Fichero: copia.cpp

#include <iostream>
using namespace std;

int main ()
{
   const int TAM_BUFFER = 10;
   char buffer[TAM_BUFFER];

   while (cin.read(buffer, TAM_BUFFER)) {
      cout.write(buffer, TAM_BUFFER);
   }
   cout.write(buffer, cin.gcount());
   return (0);
}
```

```
% copia
Este.es.un.ejemplo(ENTER)
Este.es.unde.copia.de.la(ENTER)
.ejemplo
de.copia.deentrada(ENTER)
.la
entrad(CTRL+D)a
%
```

5.6 Redirección y encauzamiento

- Al empezar la ejecución de un programa en C++ se dispone de tres flujos:
  - `cin`. E. estándar (*teclado*).
  - `cout`. S. estándar (*consola*).
  - `cerr`. S. de errores estándar (*consola*).

  siempre que se incluya `iostream`

- Los sistemas operativos proporcionan mecanismos para cambiar la entrada y salida estándar y encauzarla desde/a otros programas.
  - **Redirección de entrada**: <
  - **Redirección de salida**: >
  - **Encauzamiento** (*pipes*): |

- **Clave**:

  Puede redireccionarse la entrada siempre que el programa utilice el el objeto `cin`*, y puede redireccionarse la salida siempre que el programa utilice el objeto* `cout`*.*

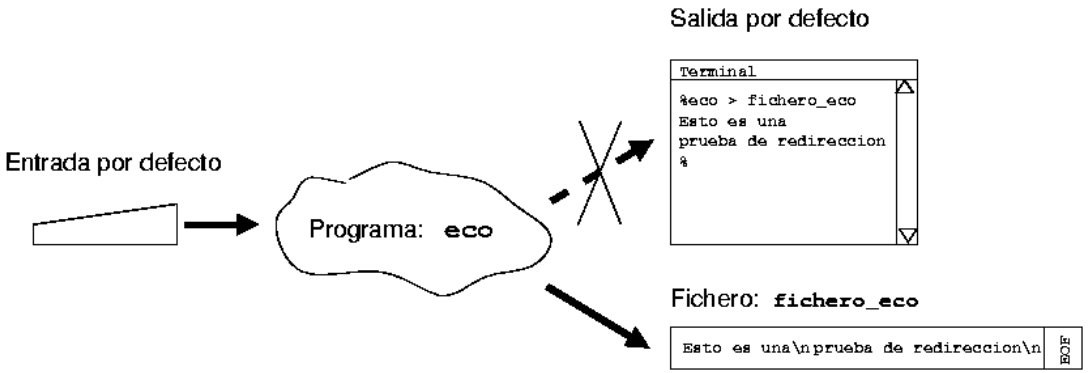
```
// Fichero: eco.cpp
#include <iostream>
using namespace std;
int main ()
{
   char c;

   while ((c = cin.get()) != EOF)
      cout.put (c);
   return (0);
}
```

```
// Fichero: cuenta_lineas.cpp
#include <iostream>
using namespace std;
int main ()
{
   char c;
   int contador = 0;

   while ((c = cin.get()) != EOF)
      if (c == '\n') contador++;
   cout << "Num. de lineas = " << contador << endl;
   return (0);
}
```

1. Redirección de salida

```
% eco > fichero_eco
Esto es una (ENTER)
prueba de redireccion (ENTER)
(CTRL+D)
```

```
% cat fichero_eco
Esto es una
prueba de redireccion
```

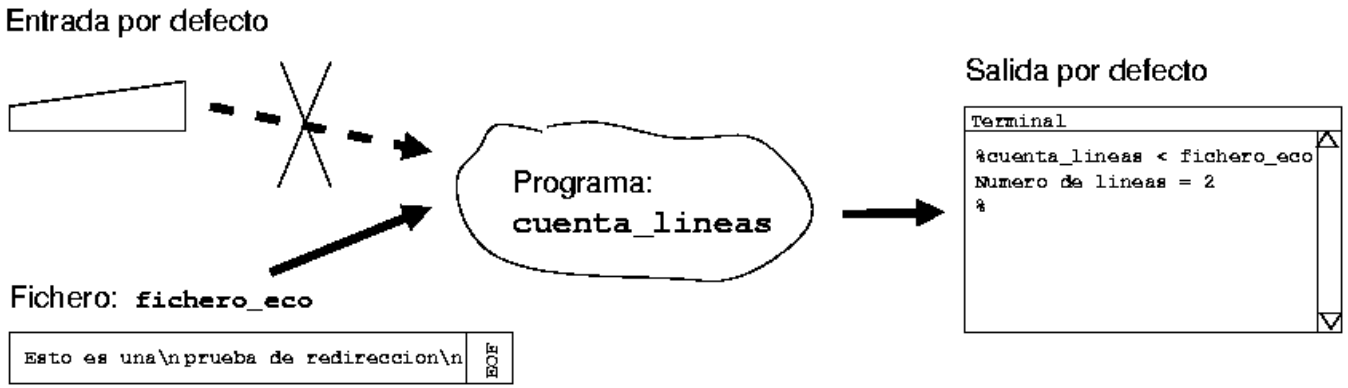


Redir. de salida: `eco > fichero_eco`

```
% cuenta_lineas > fichero_cuenta
Esto es una (ENTER)
prueba de redireccion (ENTER)
(CTRL+D)

% cat fichero_cuenta
Numero de lineas =   2
```

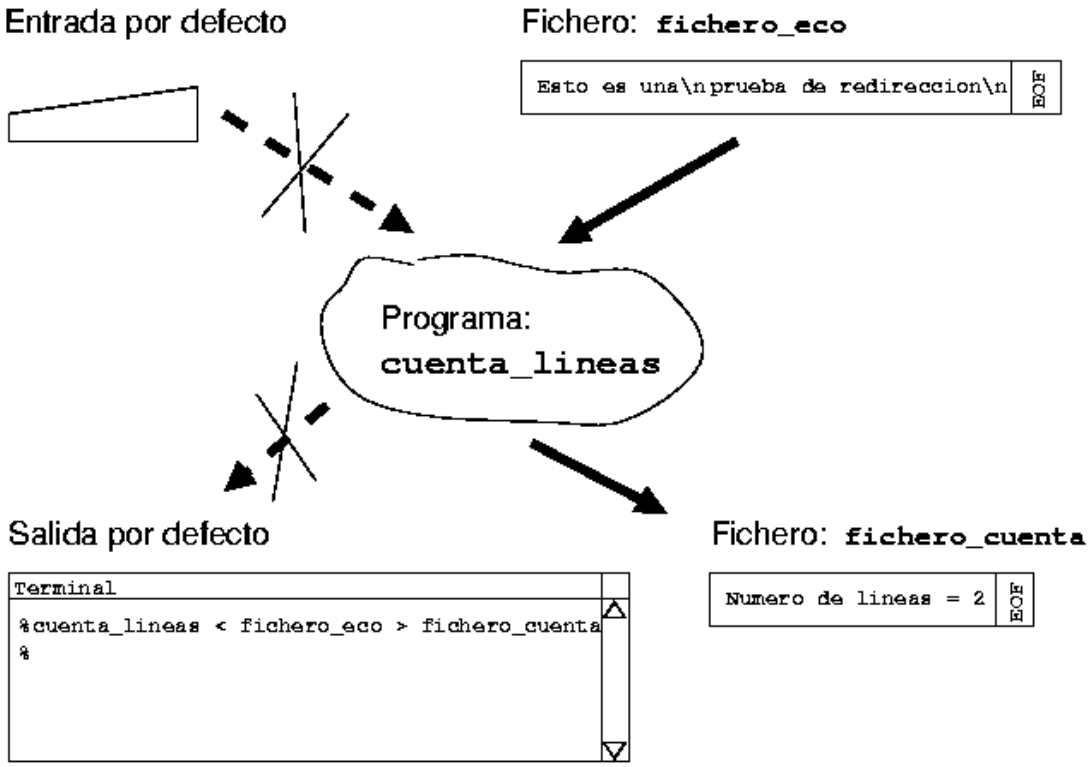
2. **Redirección de entrada**.

```
% cuenta_lineas < fichero_eco
Numero de lineas =   2
```



Redir. de entrada: `cuenta_lineas < fichero_eco`

```
% cuenta_lineas < fichero_eco > fichero_cuenta
```



Redirección de entrada y salida:

`cuenta_lineas <fichero_eco >fichero_cuenta`

3. **Encauzamiento** (*pipes*).

La salida de un programa se pasa directamente como la entrada de otro, sin ficheros intermedios.

```
% eco | cuenta_lineas
Esto es una (ENTER)
prueba de redireccion (ENTER)
(CTRL+D)
Numero de lineas =   2
```

Redirección y encauzamiento pueden combinarse:

```
% eco < fich_E | cuenta_lineas
```

```
% eco < fich_E | cuenta_lineas > fich_S
```

Como parece lógico, **¡No está permitido redireccionar una salida y encauzarla!**

```
% prog1 | prog2 | prog3 | prog4
```

```
% prog1 < fich_E | prog2 | prog3 | prog4 > fich_S
```

Otros ejemplos

```
% cuenta_lineas < fichero_eco
Num. de lineas = 2
```

⋆ Usando `tamanio` (trasp. 26):

```
% tamanio < fichero_eco
Tamanio = 34
% cuenta_lineas < fichero_eco | tamanio
Tamanio = 19
```

⋆ Usando `copia` (trasp. 27):

```
% copia < fichero_eco
Esto es una
prueba de redireccion
```

```
% eco < fichero_eco
Esto es una
prueba de redireccion
```

⋆ eco y copia pueden emplearse para copiar ficheros:

```
% eco < fichero_eco > copia_fichero_eco

% copia < copia_fichero_eco
Esto es una
prueba de redireccion

% copia < fichero_eco > copia2_fichero_eco

% eco < copia2_fichero_eco
```

```
Esto es una
prueba de redireccion

% ls -l *eco*
-rw-r--r-- 1 pepe ... 34 ... copia2_fichero_eco
-rw-r--r-- 1 pepe ... 34 ... copia_fichero_eco
-rw-r--r-- 1 pepe ... 34 ... fichero_eco

% tamanio < fichero_eco
Tamanio = 34
% tamanio < copia_fichero_eco
Tamanio = 34
% tamanio < copia2_fichero_eco
Tamanio = 34
```

5.7 Personalizar las E/S

- Mediante:

  1. Manipuladores de flujo.
  2. "Banderas" de estado del flujo.

- Recursos en `iomanip`.

- **Manipuladores**

  - Un manipulador es una función a la que se llama de una manera no tradicional. La función manipuladora, a su vez, llama a una función miembro.

- C++ proporciona *manipuladores de flujo* para realizar tareas de formato:
  - Inserción de saltos de línea: `endl`.
  - Base: `setbase()`, `dec`, `oct` y `hex`.
  - Precisión:`setprecision()` y `precision()`
  - Anchura: `setw()` y `width()`.
  - Relleno: `setfill()` y `fill()`.

- **"Banderas" de estado del flujo**.

  - Todo flujo tiene unos *indicadores de estado* que pueden activarse o desactivarse (p.e. mostrar el signo o no).
  - Activación:
    A) `cout.setf (ios::showpos);`
    B) `cout << setiosflags (ios::showpos);`
  - Desactivación:
    A) `cout.unsetf (ios::showpos);`
    B) `cout << resetiosflags (ios::showpos);`
  - Al activar una bandera se desactiva automáticamente otra que estuviera activa, si son excluyentes.

`ios::fixed`	Escribir números reales en notación de punto fijo. Se desactiva `ios::scientific`
`ios::scientific`	Escribir números reales en notación científica (`E`). Se desactiva `ios::fixed`
`ios::showpoint`	Mostrar siempre el punto decimal y los ceros a la derecha en números reales.
`ios::showpos`	Mostrar el signo.
`ios::right`	Alineación a la derecha. Se descativa `ios::left`
`ios::left`	Alineación a la izquierda. Se descativa `ios::right`
`ios::showbase`	Al mostrar un número, poner el prefijo `0` si se muestra `oct` ó `0x` si se muestra `hex`.
`ios::uppercase`	Mostrar en mayúsculas `0X` y la letra `E` en not. científica.
`ios::dec`	Los enteros deben tratarse `ios::oct` en base decimal, octal o `ios::hex` hexadecimal.

5.7.1 Manipuladores. Base

```
// Fichero: base.cpp

#include <iostream>
#include <iomanip>
using namespace std;
int main ()
{
   int n;

   cout << "\nIntroduzca un numero entero: ";
   cin >> n;

   cout << n << " en hexadecimal: " << hex << n << endl;
```

```
   cout << dec << n << " en octal: "<< oct << n << endl;
   cout << setbase(10) << n << " en dec.: "<< n << endl;

   return (0);
}

% base
Introduzca un numero entero: 26
26 en hexadecimal: 1a
26 en octal: 32
26 en dec.: 26
```

```
// Fichero: base2.cpp

#include <iostream>
#include <iomanip>
using namespace std;

int main ()
{
   int n;

   cout << "\nIntroduzca un numero entero: ";
   cin >> n;
   cout << setiosflags (ios::showbase);   // idem. con:
                          // cout.setf (ios::showbase);
   cout.setf (ios::uppercase);            // idem. con:
              // cout << setiosflags (ios::uppercase);
```

```
   cout << n << " en hexadecimal: " << hex << n << endl;
   cout << dec << n << " en octal: "<< oct << n << endl;
   cout << setbase(10) << n << " en dec.: "<< n << endl;

   cout.unsetf (ios::uppercase);          // idem. con:
            // cout << resetiosflags (ios::uppercase);

   cout << "Recuerde que " << dec << n
        << " en hexadecimal es " << hex << n << endl;
   return (0);
}

% base2
Introduzca un numero entero: 26
26 en hexadecimal: 0X1A
26 en octal: 032
26 en dec.: 26
Recuerde que 26 en hexadecimal es 0x1a
```

5.7.2 Manipuladores. Precisión

```
// Fichero: precision.cpp
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
int main ()
{
   const double v = 2.0;
   double r = sqrt(v);

   cout << setiosflags (ios::fixed);
   cout << "\nRaiz cuadrada de "<< v
        <<  " con precision 0-9.\n";
   cout << "\nPrecision con func. precision():\n";
```

```
   for (int dec=0; dec < 10; dec++) {
      cout.precision (dec);
      cout << "  " << dec << ": "
           << "|" << r  << "|" << endl;
   }
   cout << "\nPrecision con manip. setprecision():\n";
   for (int dec=0; dec < 10; dec++)
      cout << setprecision (dec) << "  " << dec << ": "
           << "|" << r  << "|" << endl;
   return (0);
}
```

```
% precision
Raiz cuadrada de 2.000000 con precision 0-9.

Precision con func. precision():
  0: |1|
  1: |1.4|
  2: |1.41|
  3: |1.414|
  4: |1.4142|
  5: |1.41421|
  6: |1.414214|
  7: |1.4142136|
  8: |1.41421356|
  9: |1.414213562|
```

```
Precision con manip. setprecision():
  0: |1|
  1: |1.4|
  2: |1.41|
  3: |1.414|
  4: |1.4142|
  5: |1.41421|
  6: |1.414214|
  7: |1.4142136|
  8: |1.41421356|
  9: |1.414213562|
```

5.7.3 Manipuladores. Ancho

```
// Fichero: ancho.cpp

#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

int main ()
{  const double v = 2.0;
   double r = sqrt(v);

   cout << setiosflags (ios::fixed);
   cout << "\nRaiz cuadrada de " << v <<  " con precision 0-9.\n";
   cout << "\nPrecision con func. " << "precision() y width():\n";
```

```
   for (int dec=0; dec < 10; dec++) {
      cout.width(5);
      cout <<  dec << ":" << "|" ;
      cout.precision (dec);
      cout.width(11);
      cout  << r <<  "|" << endl;
   }
   cout << "\nPrecision con manip. << "setprecision() y setw():\n";

   for (int dec=0; dec < 10; dec++) {
      cout << setw (5) << dec << ":" << "|" ;
      cout << setprecision (dec) << setw (11) << r << "|" << endl;
   }
   return (0);
}
```

```
% ancho

Raiz cuadrada de 2.000000 con precision 0-9.

Precision con func. precision() y width():
    0:|           1|
    1:|         1.4|
    2:|        1.41|
    3:|       1.414|
    4:|      1.4142|
    5:|     1.41421|
    6:|    1.414214|
    7:|   1.4142136|
    8:|  1.41421356|
    9:|1.414213562|
```

```
Precision con manip. setprecision() y setw():
    0:|          1|
    1:|        1.4|
    2:|       1.41|
    3:|      1.414|
    4:|     1.4142|
    5:|    1.41421|
    6:|   1.414214|
    7:|  1.4142136|
    8:| 1.41421356|
    9:|1.414213562|
```

5.7.4 Manipuladores. Relleno

```
// Fichero: relleno.cpp
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

int main ()
{  const double v = 2.0;
   double r = sqrt(v);

   cout << setiosflags (ios::fixed);
   cout << "\nRaiz cuadrada de " << v <<  " con precision 0-9.\n";
   cout << "\nRelleno con func. fill():\n";
```

```
for (int dec=0; dec < 10; dec++) {
   cout.width(5);
   cout.fill ('.');
   cout <<  dec << ":" << "|" ;
   cout.precision (dec);
   cout.width(15);
   cout.fill ('*');
   cout  << r <<  "|" << endl;
}
cout << "\nRelleno con manip. setfill():\n";

for (int dec=0; dec < 10; dec++) {
   cout << setw (5) << setfill('.') << dec <<":"<<"|";
   cout << setprecision (dec) << setw (15)
        << setfill ('*') << r << "|" << endl;
}
```

```
   return (0);
}

% relleno
Raiz cuadrada de 2.000000 con precision 0-9.

Relleno con func. fill():
....0:|**************1|
....1:|*************1.4|
....2:|************1.41|
....3:|***********1.414|
....4:|**********1.4142|
....5:|*********1.41421|
....6:|********1.414214|
....7:|*******1.4142136|
....8:|******1.41421356|
....9:|*****1.414213562|
```

```
Relleno con manip. setfill():
....0:|**************1|
....1:|*************1.4|
....2:|************1.41|
....3:|***********1.414|
....4:|**********1.4142|
....5:|*********1.41421|
....6:|********1.414214|
....7:|*******1.4142136|
....8:|******1.41421356|
....9:|*****1.414213562|
```

```
// Fichero: relleno2.cpp
#include <iostream>
#include <iomanip>
using namespace std;

int main ()
{  const int TAM = 5;
   float v[TAM] = {10.5, 6.33, 334.5, 55.7643, -5.3};

   cout << setiosflags (ios::fixed) << "\nCon funciones miembro:\n";
   for (int n=0; n < TAM; n++) {
      cout.setf (ios::left);
      cout.width(3);
      cout.fill ('.');
      cout << n << " : ";
      cout.unsetf (ios::left);
```

```
      cout.setf (ios::right);
      cout.precision (5);
      cout.width(15);
      cout.fill ('*');
      cout << v[n] << endl;
      cout.unsetf (ios::right);
   }
   cout << "\nCon manipuladores:\n";

   for (int n=0; n < TAM; n++) {
      cout << setiosflags (ios::right) << setw (3) << setfill('.') << n << " : ";
      cout << resetiosflags (ios::right) << setiosflags (ios::left)
           << setprecision (5) << setw (15) << setfill ('*') << v[n] << endl;
      cout << resetiosflags (ios::left);
   }
   return (0);
}
```

```
% relleno2
Con funciones miembro:
0.. : *******10.50000
1.. : ********6.33000
2.. : ******334.50000
3.. : *******55.76430
4.. : *******-5.30000

Con manipuladores:
..0 : 10.50000*******
..1 : 6.33000********
..2 : 334.50000******
..3 : 55.76430*******
..4 : -5.30000*******
```

```
// Fichero: tabla.cpp
#include <iostream>
#include <iomanip>
using namespace std;

int main ()
{
   const int NUM = 34;

   cout.setf (ios::uppercase);
   cout.setf (ios::right);

   cout << setw(10) << "DEC"<< setw(5) << "OCT" << setw (5) << "HEX"<< endl;

   for (int n=0; n<NUM; n++) {
      cout.fill(' ');
      cout.setf (ios::dec);
```

```
      cout << setw (3) << n << " :";
      cout.unsetf (ios::dec);

      cout.fill('.');
      cout << setw (5) << setiosflags (ios::dec) << n;
      cout.unsetf (ios::dec);

      cout << setw (5) << setiosflags (ios::oct) << n;
      cout.unsetf (ios::oct);

      cout << setw (5) << setiosflags (ios::hex) << n;
      cout.unsetf (ios::hex);
      cout << endl;
   }
   return (0);
}
% tabla
```

```
      DEC  OCT  HEX
 0 :....0....0....0
 1 :....1....1....1
 2 :....2....2....2
 3 :....3....3....3
 4 :....4....4....4
 5 :....5....5....5
 6 :....6....6....6
 7 :....7....7....7
 8 :....8...10....8
 9 :....9...11....9
10 :...10...12....A
11 :...11...13....B
12 :...12...14....C
13 :...13...15....D
14 :...14...16....E
15 :...15...17....F
```

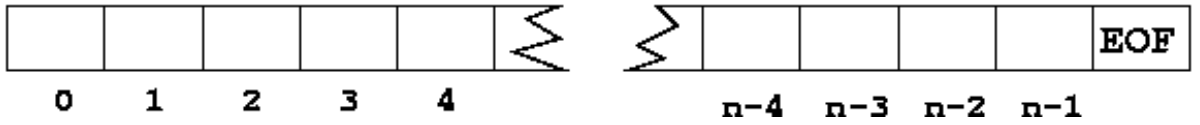
```
16 :...16...20...10
17 :...17...21...11
18 :...18...22...12
19 :...19...23...13
21 :...21...25...15
22 :...22...26...16
23 :...23...27...17
24 :...24...30...18
25 :...25...31...19
26 :...26...32...1A
27 :...27...33...1B
28 :...28...34...1C
29 :...29...35...1D
30 :...30...36...1E
31 :...31...37...1F
32 :...32...40...20
33 :...33...41...21
```

5.8 Ficheros. Introducción

- **Motivación**: Conservar datos de forma *permanente*: accesibles para diferentes programas y ejecuciones.

- Los datos se organizan en **ficheros**, que se ubican en dispositivos de almacenamiento masivo.

- C++ ve a cada fichero como una *secuencia de bytes*, terminados por `EOF`.



Representación de un fichero de n bytes

- El procesamiento de ficheros tiene muchas similitudes con la gestión de E/S. (Recordar la jerarquía de clases)

- Las operaciones a nivel físico las realiza el sistema operativo $\Rightarrow$ el programador no debe preocuparse de cómo se hacen.

5.9 Apertura y cierre de ficheros

- Procedimiento para utilizar un fichero en un programa en C++:

  1. En primer lugar, se ***abre***,
  2. después se procesa y,
  3. finalmente, cuando se haya terminado su procesamiento, se ***cierra***.

- **Apertura de ficheros**.

  En realidad, la apertura consiste en *asociar un flujo a un fichero*:

A) Si el fichero se va a utilizar para lectura, se le asocia un **flujo de entrada** (un objeto de la clase `ifstream`).

B) Si el fichero se va a utilizar para escritura, se le asocia un **flujo de salida** (un objeto de la clase `ofstream`).

$\Rightarrow$ Incluir el fichero de cabecera `fstream`

Métodos open().

- Apertura de un fichero para lectura:

```
#include <fstream>
....
// Declaracion del flujo de entrada "fi"

ifstream fi;

// Asociar el flujo de entrada "fi"
// al fich. "fichero_eco"

fi.open ("fichero_eco");
....
// Compacta: ifstream fi ("fichero_eco");
```

- Apertura de un fichero para escritura:

```
#include <fstream>
....
// Declaracion del flujo de salida "fo"

ofstream fo;

// Asociar el flujo de salida "fo"
// al fich. "copia_fichero_eco"

fo.open ("copia_fichero_eco");
....
// Compacta: ofstream fo ("copia_fichero_eco");
```

- **Cierre de ficheros**.

  En realidad, el cierre consiste en *desasociar un flujo de un fichero*.

  Una vez cerrado el fichero, el flujo puede conectarse a otro fichero (abriéndolo convenientemente).

  Métodos `close()`.

```
// Desasociar el flujo de entrada "fi" del fich. "fichero_eco"
...
fi.close ();
...
// Desasociar el flujo de salida "fo" del fich. "copia_fichero_eco"
...
fo.close ();
...
```

Una vez cerrado el fichero, no se puede volver a usar a no ser que se abra de nuevo (no hay flujo de comunicación entre el fichero y memoria).

Es importante cerrar un fichero:

1. Asegura que el archivo no quedará dañado (inconsistencia física).
2. Se libera el buffer: los datos que se escriben se copian realmente en el disco.

3. Pone el carácter `EOF` en el flujo de S.

- **Procesamiento**.

  Clave: **Herencia**

  1. `ifstream` deriva de `istream`
     Un objeto de clase `ifstream` tiene la misma funcionalidad que otro de la clase `istream`, en particular, que `cin`.

  2. `ofstream` deriva de `ostream`
     Un objeto de clase `ofstream` tiene la misma funcionalidad que otro de la clase `ostream`, en particular, que `cout`.

```
// Fichero:  escribe_letra_A.cpp

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

int main ()
{
   ofstream fo;

   fo.open ("letra_A"); // Apertura
   fo << 'A' << endl; // Procesamiento
   fo.close(); // Cierre

   return (0);
}
```

```
% escribe_letra_A

% cat letra_A
A

% ls -l
...
-rw-r--r--  1 pepe  alumnos  2 Mar 15 11:15 letra_A
```

```
// Fichero:  demo_out.cpp
#include <fstream>
using namespace std;

int main ()
{  int n = 5;
   float r = 7.33;
   char  *cad = "Hola";
   int   *pi  = &n;
   float *pf  = &r;

   ofstream fo;
   fo.open ("sal_demo_out");
   fo << "Literal de cadena" << endl;
   fo << n << endl;
   fo << r << endl;
   fo << *pi << "  "<< pi << endl;
   fo << *pf << "  "<< pf << endl;
```

```
   fo << cad << "  "<< static_cast <void *> (cad)<< endl;
   fo.close ();
   return (0);
}

% demo_out
% ls -l
...
-rw-r--r--  1 pepe  alumnos 72 abr 17 10:42 sal_demo_out
...

% cat sal_demo_out
Literal de cadena
5
7.33
5  0xbffff82c
7.33  0xbffff828
Hola  0x8048b18
```

```
// Fichero:  demo_in.cpp

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

int main ()
{
   int n;
   float r;
   char cad[50];

   ifstream fi ("ent_demo_in");
   fi >> n >> r >> cad;
   fi.close ();

   cout << "\nValores leidos: \n";
```

```
   cout << "   Entero: " << n << endl;
   cout << "   Real: " << r << endl;
   cout << "   Cadena:  " << cad << endl << endl;
   return (0);
}

% cat ent_demo_in
5
7.33
Una cadena

% demo_in
Valores leidos:
   Entero: 5
   Real: 7.33
   Cadena:  Una
```

```
// Fichero:  escribe_letra_B.cpp
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;

int main ()
{
   ofstream fo;
   ifstream fi;
   char c;

   fo.open ("letra_B"); // Apertura
   fo << 'B' << endl; // Procesamiento
   fo.close(); // Cierre
   fi.open ("letra_B"); // Apertura
   fi >> c; // Procesamiento
```

```
   fi.close(); // Cierre
   cout << "Caracter leido: " << c << endl;
   return (0);
}

%  escribe_letra_B
Caracter leido: B

% cat letra_B
B

% ls -l
...
-rw-r--r--  1 pepe  alumnos  2 Mar 15 12:06 letra_B
...
```

Comprobar el resultado de `open()` !!

- En modo E (**lectura**). Puede fallar si el fichero no existe o no se tiene permiso para lectura.

```
fi.open ("letra_B"); // Apertura
if (fi.fail()) {
   cerr << "Error: no pudo abrirse letra_B\n";
   exit (1);
}
```

Alternativamente:

`!fi` en lugar de `fi.fail()`

- En modo S (**escritura**). Puede fallar si no hay espacio disponible o no se tiene permiso para escritura.

```
fo.open ("letra_B"); // Apertura
if (fo.fail()) {
   cerr << "Error: no pudo crearse letra_B\n";
   exit (1);
}
```

Alternativamente:

`!fo` en lugar de `fo.fail()`

```
// Fichero:  escribe_letra_A.cpp
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;

int main ()
{  ofstream fo;
   fo.open ("letra_A"); // Apertura
                        // Comprobar si se ha podido crear el fichero "letra_A"
   if (fo.fail()) { // Abortar la ejecucion
      cerr << "Error: no pudo crearse letra_A\n";
      exit (1);
   }
   fo << 'A' << endl; // Procesamiento
   fo.close(); // Cierre
   return (0);
}
```

```
// Fichero:  escribe_letra_B.cpp

#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
int main ()
{
   ofstream fo;
   ifstream fi;
   char c;

   fo.open ("letra_B"); // Apertura
   // Comprobar si se ha podido crear "letra_B"
   if (!fo) {
      cerr << "Error: no pudo crearse letra_B\n";
      exit (1);
```

```
   }
   fo << 'B' << endl; // Procesamiento
   fo.close(); // Cierre

   fi.open ("letra_B"); // Apertura
   // Comprobar si se ha podido abrir "letra_B"
   if (!fi) {
      cerr << "Error: no pudo abrirse letra_B\n";
      exit (1);
   }
   fi >> c; // Procesamiento
   fi.close(); // Cierre

   cout << "Caracter leido: " << c << endl;
   return (0);
}
```

```
// Fichero:  suma_int_file.cpp

#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;
int main ()
{
   ifstream fi;
   int n, cont=0, sum=0;

   fi.open ("datos");
   if (!fi) {
      cerr << "Error: no pudo abrirse datos\n";
      exit (1);
   }
```

```
   cout<< "\n(Suma acumulada = "<<setw(5)<< sum<< "). ";
   while (fi >> n) {
      cout <<"Valor leido num. "<<setw(3)<<cont+1<<" : "
           << setw(3) << n << endl;
      sum += n;
      cont++;
      cout <<"(Suma acumulada = "<<setw(5)<< sum<< "). ";
   }
   cout << "\nLa suma total de los " << cont;
   cout << " enteros leidos es " << sum;
   cout << endl;

   fi.close();
   return (0);
}
```

```
% cat datos
2
44
3
8

% suma_int_file
(Suma acumulada =     0). Valor leido num.   1 :   2
(Suma acumulada =     2). Valor leido num.   2 :  44
(Suma acumulada =    46). Valor leido num.   3 :   3
(Suma acumulada =    49). Valor leido num.   4 :   8
(Suma acumulada =    57).
La suma total de los 4 enteros leidos es 57
```

5.10 Nombres de fichero

- `open()` recibe como argumento un `char *`
- El argumento se interpreta como la dirección de memoria de una **cadena clásica**:

  una sucesión de caracteres delimitada por el carácter nulo (`'\0'`*) que se interpreta como el delimitador de fin de cadena*.

  $\Longrightarrow$ **no es un objeto** `string`.
- En nuestra discusión supondremos que los nombres de fichero no contienen espacios en blanco.

1. Lectura del nombre con el operador >>

```
......
ifstream if;
char nombre[255];
......
cout << "Nombre del fichero: \n";
cin  >> nombre;
......
if.open (nombre);
......
```

```
// Fichero: type.cpp

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

int main ()
{
   ifstream fuente;
   char nombre[255];
   char c;

   cout << "\nNombre del fichero: ";
   cin  >> nombre;

   fuente.open (nombre);
```

```
    if (!fuente) {
       cerr << "Error: no puedo abrir "
            << nombre << endl;
       exit (1);
    }
    while ((c = fuente.get()) != EOF)
       cout.put (c);

    fuente.close();

    return (0);
}
```

2. Tomar el nombre de la línea de órdenes.

```
// Fichero: type2.cpp
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;

int main (int argc, char **argv)
{
   ifstream f;
   char c;

   if (argc != 2) {
      cerr << "Error: Num. de params. erroneo\n";
      cerr << "Uso: "<< argv[0]<<" <fichero>\n";
      exit (1);
```

```
    }
    f.open (argv[1]);
    if (!f) {
       cerr << "Error: no puedo abrir "
            << argv[1] << endl;
       exit (1);
    }
    while ((c = f.get()) != EOF)
       cout.put (c);
    f.close();
    return (0);
}
```

3. Tomar el nombre de un objeto `string`.

```
// Fichero: type3.cpp
#include <fstream>
#include <string>
#include <cstdlib>
using namespace std;

int main (int argc, char **argv)
{
   ifstream f;
   string nombre;
   char c;

   cout << "\nNombre del fichero: ";
   cin  >> nombre;
```

```
    f.open (nombre.c_str());
    if (!f) {
       cerr << "Error: no puedo abrir "
            << nombre << endl;
       exit (1);
    }
    while ((c = f.get()) != EOF)
       cout.put (c);

    f.close();
    return (0);
}
```

Ejemplo

Comprobar si existe un fichero:

```
/*****************************************/

bool ExisteFichero (const string &nombre)
{
   ifstream fichero;
   bool problema;

   fichero.open (nombre.c_str());
   problema = fichero.fail();
   if (!problema) fichero.close();
   return ((problema) ? false : true);
}
/*****************************************/
```

```
/*****************************************/

bool ExisteFichero (char *nombre)
{
   ifstream fichero;
   bool problema;

   fichero.open (nombre);
   problema = fichero.fail();
   if (!problema) fichero.close();
   return ((problema) ? false : true);
}
/*****************************************/
```

5.11 Flujos como argumentos

- Un flujo puede ser argumento de una función.
- Restricción: debe pasarse por **referencia**.

Ejemplo: "Asear" un fichero

sucio.dat

```
     37.10       -4.8765
3.545     -7.554   22.0


6.77777     88.33459
-3.2342424e2
```

limpio.dat

```
 +37.10000
  -4.87650
  +3.54500
  -7.55400
 +22.00000
  +6.77777
 +88.33459
-323.42424
```

```
// Fichero: asear.cpp
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdlib>
using namespace std;

int main (int argc, char **argv)
{
   ifstream fi;
   ofstream fo;
   void asear (ifstream & entrada, ofstream & salida, int num_dec, int ancho);

   if (argc != 3) {
      cerr << "Error: Num. de params. incorrecto\n";
      cerr << "Uso: "<< argv[0]<< " <fichE> <fichS>\n";
      exit (1);
```

```
   }
   fi.open (argv[1]);
   if (!fi) {
      cerr << "Error: no puedo abrir "<<argv[1]<<endl;
      exit (1);
   }
   fo.open (argv[2]);
   if (!fo) {
      cerr << "Error: no puedo crear "<<argv[2]<<endl;
      exit (1);
   }
   asear (fi, fo, 5, 12);

   fi.close();
   fo.close();
   return (0);
}
```

```
void asear (ifstream & entrada, ofstream & salida, int num_dec, int ancho )
{  double num;

   salida.setf (ios::fixed);
   salida.setf (ios::showpoint);
   salida.setf (ios::showpos);
   salida.precision (num_dec);
   cout.setf (ios::fixed);
   cout.setf (ios::showpoint);
   cout.setf (ios::showpos);
   cout.precision (num_dec);
   while (entrada >> num) {
      cout << setw (ancho) << num << endl;
      salida << setw (ancho) << num << endl;
   }
}
```

Este programa es similar al `wc` de Linux.

```
// Fichero: cuenta_cosas.cpp
#include <fstream>
#include <stdlib>
#include <cctype>
#include <iomanip>
using namespace std;

void cuenta (ifstream & f, int &l, int& p, int& c);

int main (int argc, char **argv)
{
   ifstream fi;
   int nlineas, npalabras, ncaracteres;
   int tot_lineas,tot_palabras,tot_caracteres;
   int i, num_args = argc;
```

```
if (argc == 1) {
   cerr << "Error: Num. de params. erroneo\n";
   cerr << "Uso: " << argv[0]<< " <fich1> ";
   cerr << "[<fich2>...<fichn>]\n\n";
   exit (1);
}
tot_lineas = tot_palabras = tot_caracteres = 0;
for (i=1; i<=num_args-1; i++) {
   fi.open (argv[i]);
   if (!fi) {
      cerr << "Error: no puedo abrir "
           << argv[i] << endl;
   }
   else {
      cuenta (fi, nlineas, npalabras, ncaracteres);
```

```
         fi.close ();
         tot_caracteres += ncaracteres;
         tot_lineas += nlineas;
         tot_palabras += npalabras;

         cout << setw(7) << nlineas << setw(8) << npalabras
              << setw(8) << ncaracteres << " " << argv[i] << endl;
      } // else de: if (!fi)

   } //for (i=1; i<=num_args-1; i++)

   if (num_args > 2)
      cout << setw(7) << tot_lineas << setw(8) << tot_palabras
           << setw(8) << tot_caracteres << " total\n";
   return (0);
}
/*************************************************/
```

```
void cuenta (ifstream& f, int& nl, int& np, int& nc)
{
   bool en_palabra=true;
   char c;

   nl = np = nc = 0;

   while ((c = f.get ()) != EOF) {

      nc++;

      if (isspace(c)) {

         if (en_palabra) {
            np++;
            en_palabra = false;
```

```
         }
         if (c == '\n') nl++;
      }
      else // no es separador

         if (!en_palabra) en_palabra=true;

   } //  while ((c = fi.get ()) != EOF) {
}
```

Ejemplo

Presentaremos dos versiones de un programa (similar al `cat`) que pone en `cout` (siempre) lo que recibe del flujo de entrada.

Versión 1: Copia en `cout`:

1. Lo que recibe de `cin` (hace el eco de la entrada) cuando el programa se llama sin argumentos.

```
% mi_cat
Esto es una prueba(ENTER)
Esto es una prueba
de mi_cat(ENTER)
de mi_cat
(CTRL+D)
```

2. Lo que recibe de un fichero, cuando el programa se llama con un argumento: el nombre del fichero.

```
% mi_cat texto
```
Esto es una prueba
de mi_cat

```
// Fichero: mi_cat.cpp

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

int main (int argc, char **argv)
{
   ifstream fi;
```

```
void copiar (ostream & salida, istream & entrada);

if (argc > 2) {
   cerr << "Error: Num. de params. erroneo\n";
   cerr << "Uso: "<<argv[0]<<" [<ficheroE>]\n";
   exit (1);
}
if (argc == 1)
   copiar (cout, cin);
else {
   fi.open (argv[1]);
   if (!fi) {
      cerr << "Error: no puedo abrir "
           << argv[1] << endl;
      exit (1);
   }
```

```
      copiar (cout, fi);

      fi.close();
   }
   return (0);
}

/************************************************/

void copiar (ostream & salida, istream & entrada)
{
   int c;

   while ((c = entrada.get()) != EOF)
      salida.put (c);
}
/************************************************/
```

Versión 2:

Copia en `cout`:

1. Como en la versión anterior, lo que recibe de `cin` (hace el eco de la entrada) cuando el programa se llama sin argumentos.

2. Lo que recibe de una lista (indeterminada) de ficheros, cuando el programa se llama con al menos, un argumento: cada argumento es el nombre de un fichero.

```
// Fichero: mi_cat2.cpp

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

```
int main (int argc, char **argv)
{
   ifstream fi;

   void copiar (ostream & salida, istream & entrada);

   if (argc == 1)
      copiar (cout, cin);
   else {
      for (int i=1; i<argc; i++) {

         fi.open (argv[i]);
         if (!fi) {
            cerr << "Error: no puedo abrir "
                 << argv[i] << endl;
         }
```

```
         copiar (cout, fi);
         fi.close();
      } // for
   }
   return (0);
}

/************************************************/

void copiar (ostream & salida, istream & entrada)
{
   int c;

   while ((c = entrada.get()) != EOF)
      salida.put (c);
}
/************************************************/
```

```
% mi_cat2 letra_A pp letra_B
A
Error: no puedo abrir pp
B

% mi_cat2 letra_A pp letra_B > rdo
Error: no puedo abrir pp

% mi_cat2 rdo
A
B
```

IMPORTANTE:

1. Si una función toma un flujo de entrada como argumento y queremos que sea `cin` en algunos casos y un flujo asociado a un fichero de entrada en otros, se empleará un parámetro formal de tipo *referencia a* `istream`.

2. Si una función toma un flujo de salida como argumento y queremos que sea `cout` en algunos casos y un flujo asociado a un fichero de salida en otros, se empleará un parámetro formal de tipo *referencia a* `ostream`.

3. Si algún argumento es siempre un flujo asociado a un fichero de salida o de entrada, basta con emplear un parámetro formal de tipo*referencia a* `ofstream` o *referencia a* `ifstream`.

```
// Fichero: espaciado.cpp
#include <fstream>
#include <cstdlib>
using namespace std;

int main (int argc, char **argv)
{
   void espaciado (ofstream & s, ifstream & e, int n);

   ifstream fi;
   ofstream fo;
   int n;

   if ((argc < 3) || (argc > 4)) {
      cerr << "Error: Num. de params. erroneo\n";
      cerr << "Uso: " << argv[0]<< "<fichE> <fichS> ";
      cerr << "[<lin>]\n";
```

```
   exit (1);
}
if (argc == 3) // No se da valor a <lin>
   n = 0; // valor por defecto
else {
   n = atoi(argv[3]);
   if (n < 0) {
      cerr << "Error: <lin> debe ser >= 0\n";
      exit (1);
   }
}

fi.open (argv[1]);
if (!fi) {
   cerr << "Error: no puedo abrir "
        << argv[1] << endl;
   exit (1);
```

```
   }

   fo.open (argv[2]);
   if (!fo) {
      cerr << "Error: no puedo crear "
           << argv[2] << endl;
      exit (1);
   }

   espaciado (fo, fi, n);

   fi.close ();
   fo.close ();

   return (0);
}
```

```
/*************************************************/

void espaciado (ofstream & s, ifstream & e, int n)
{
   int c;

   while ((c = e.get()) != EOF) {
      s.put (c);
      if (c == '\n')
         for (int i=1; i<=n; i++)
            s.put (c);
   } // while
}
/*************************************************/

% espaciado limpio.dat limpio_def.dat
% cat limpio_def.dat
   +37.10000
```

```
    -4.87650
    +3.54500
    -7.55400
   +22.00000
    +6.77777
   +88.33459
  -323.42424
% espaciado limpio.dat limpio_2.dat 2
% cat limpio_2.dat
   +37.10000


    -4.87650


    +3.54500
....
```

5.12 La función `eof()`

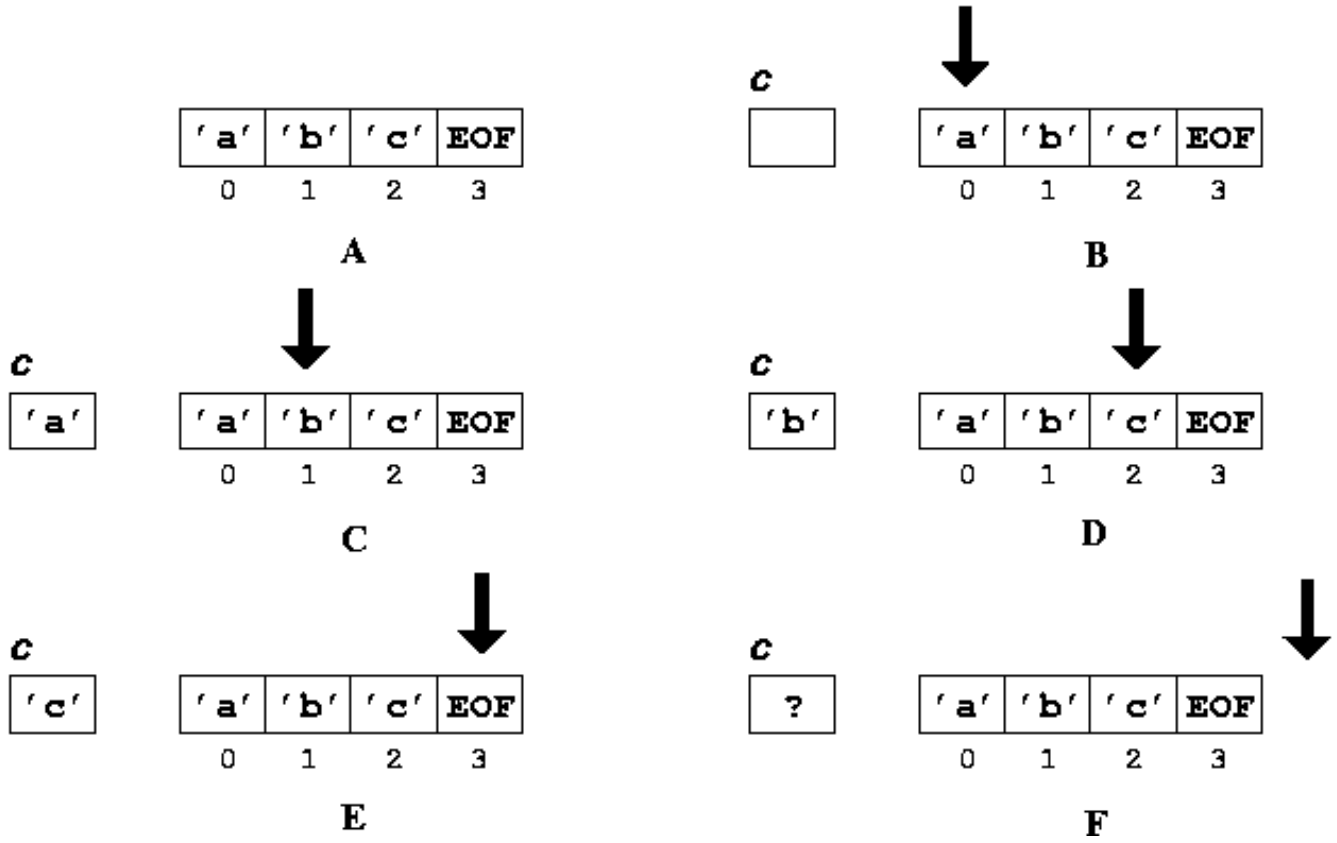
- La función `eof()` puede aplicarse a cualquier flujo de entrada, y en particular, a objetos `ifstream`.
- Utilizar `eof()` resulta la estrategia más adecuada para comprobar si se ha alcanzado el fin de un fichero.
- La estrategia de leer carácter a carácter y comprobar si se ha leido el carácter `EOF` *sólo es válida para ficheros de texto*.
- `eof()` devuelve `true` cuando se intenta leer más allá del fin del fichero:

Debe emplearse la lectura adelantada.

Ejemplo

En los ficheros `mi_cat.cpp` y `mi_cat2.cpp`, sustituir la función `copiar()` por:

```
void copiar2 (ostream & salida, istream & entrada)
{
   int c;

   c = entrada.get(); // Lectura adelantada

   while (!entrada.eof()) {
      salida.put (c);
      c = entrada.get();
   }
}
```



A) Contenido de `prueba`. B) Efecto de la apertura C) Lectura (adelantada) del primer carácter. D) Lectura del segundo E) Lectura del tercer carácter. F) Intento de lectura: se sobrepasa `EOF`

```
//Fichero: mi_cp.cpp
#include <fstream>
#include <cstdlib>
using namespace std;

int main (int argc, char **argv)
{
   void CopiaFich (ofstream & sal, ifstream & ent);

   ifstream fi;
   ofstream fo;

   if (argc != 3) {
      cerr << "Error: Num. de params. erroneo\n";
      cerr << "Uso: " << argv[0]<< " <fichE> <fichS>\n";
      exit (1);
   }
```

```
   fi.open (argv[1]);
   if (!fi) {
      cerr << "Error: no puedo abrir "
           << argv[1] << endl;
      exit (1);
   }
   fo.open (argv[2]);
   if (!fo) {
      cerr << "Error: no puedo crear "
           << argv[2] << endl;
      exit (1);
   }
   CopiaFich (fo, fi);
   fi.close ();
   fo.close ();
   return (0);
}
```

```
/*************************************************/

void CopiaFich (ofstream & sal, ifstream & ent)
{
   int c;

   c = ent.get(); // Lectura adelantada

   while (!ent.eof()) {
      sal.put (c);
      c = ent.get();
   }
}
/*************************************************/
```

5.13 Ficheros binarios

- Se considera que el fichero que se va a procesar se va a tratar a nivel de *bytes*.
- Flujo binario: consiste en una sucesión de bytes terminado por la marca de fin de fichero.
- Más general que los flujos de texto.

Ejemplo

Supongamos una aplicación que debe guardar en un fichero una serie de n números enteros positivos (representados como `unsigned int`). Supongamos que $n = 5$ y se genera la siguiente secuencia:

$$12345, \quad 34, \quad 4567345, \quad 8845, \quad 98567485.$$

Esta serie puede almacenarse en un fichero en formato texto o binario:

1. **Texto**. Criterio: almacenar cada entero en una línea del fichero o separados por un espacio o tabulador (indiferente).

   a) Si suponemos que `sizeof(unsigned int)` es 4, el máximo entero sin signo que puede utilizarse es 4294967295 (ver en `climits` la definición `#define UINT_MAX  4294967295`).

   b) Si los datos se generan de forma uniforme podemos tener valores desde 1 a 10 cifras, con la misma probabilidad *a priori*.
   Cada número ocupará tanto espacio como cifras tenga, por lo que el tamaño del fichero será la suma del número de total de cifras de los n números y del número de separadores.

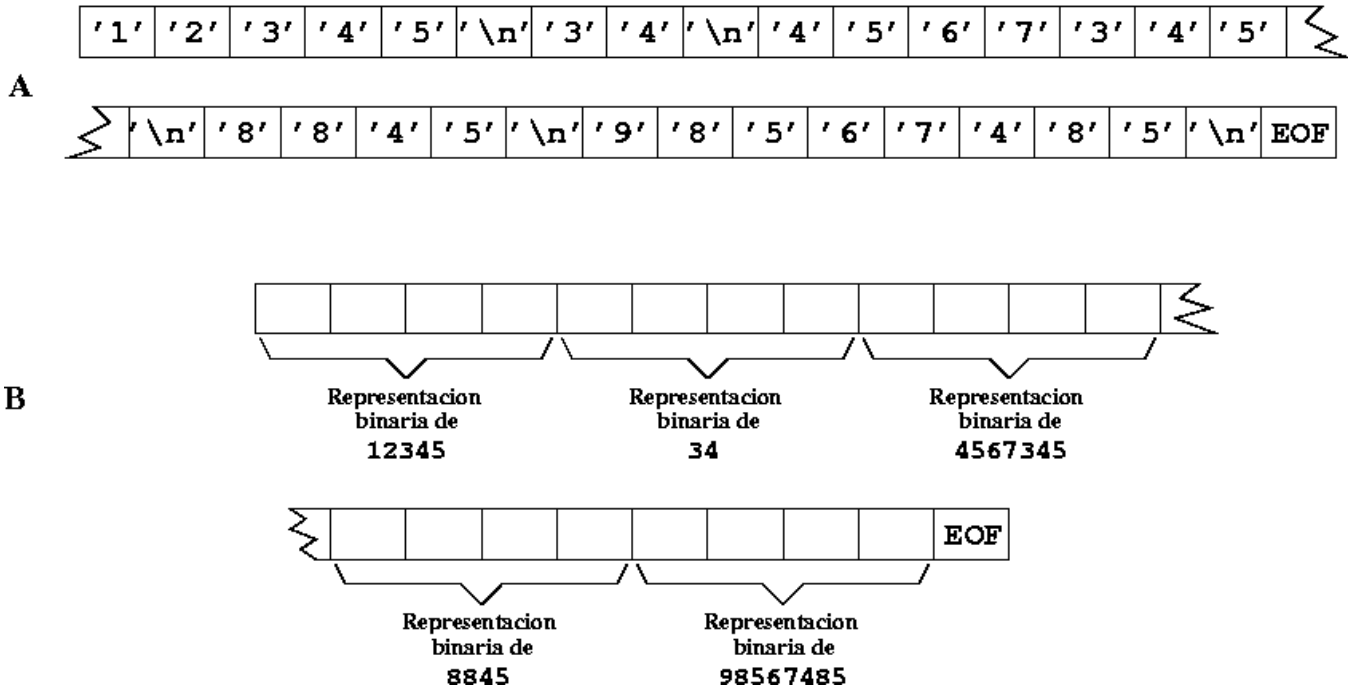
   c) El tamaño del fichero será de **31 bytes**:
   26 cifras y 5 separadores (figura A).

2. **Binario**. Cada valor se almacena según su *representación interna* (una copia exacta de como está en memoria).

   a) Cada valor se almacenaría exactamente con 4 bytes.

   b) No se necesitan separadores.

   c) El tamaño del fichero será de **20 bytes**:
   4 bytes/dato $\times$ 5 datos (figura B).



A) Representación en formato *texto* B) En formato *binario*

Muy importante: Salvo en circunstancias muy extrañas (en términos de probabilidad) la adopción del formato binario para estos ficheros de datos resulta más ventajosa.

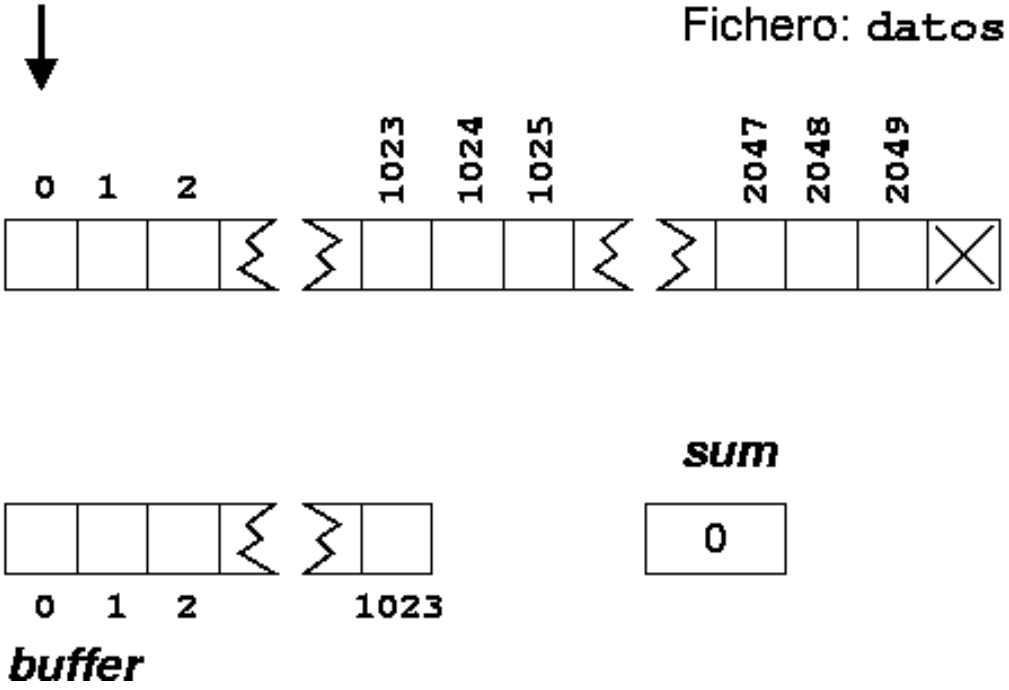
- En Linux, no se requiere especificar nada para trabajar en *modo binario* con un fichero.

- Emplear las funciones `read()` y `write()`.

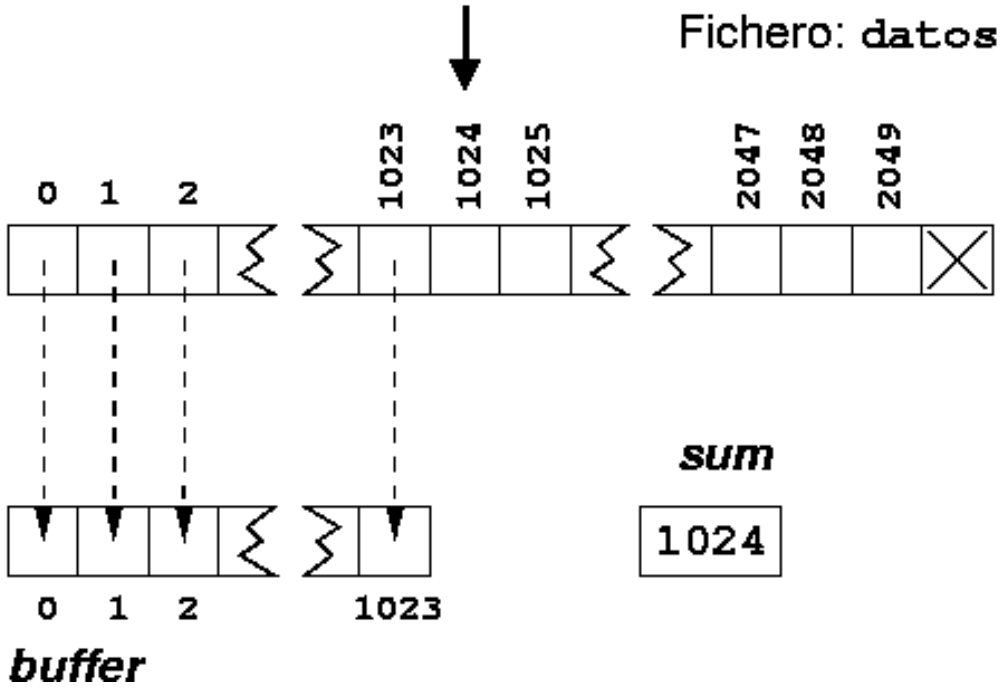
```
// Fichero: tamanio2.cpp
#include <fstream>
#include <iomanip>
#include <cstdlib>
using namespace std;

int main (int argc, char **argv)
{  const int TAM_BUFFER = 1024;
   unsigned char buffer [TAM_BUFFER];
   ifstream fi;
   int sum = 0;
   if (argc != 2) {
      cerr << "Error: Num. de params. erroneo\n";
      cerr << "Uso: " << argv[0]<< " <fich>\n";
      exit (1);
   }
   fi.open (argv[1]);
```

```
   if (!fi) {
      cerr << "Error: no puedo abrir "
           << argv[1] << endl;
      exit (1);
   }
   while (fi.read(buffer, TAM_BUFFER)) {
      sum += TAM_BUFFER;
   }
   sum += fi.gcount();
   fi.close ();
   cout.setf (ios::fixed);
   cout << "\nFichero: " << argv[1];
   cout << setw(10) << sum << " bytes, ";
   cout << setprecision(2) << setw(8) << sum/1024.0
        << " Kbytes\n\n";
   return (0);
}
```

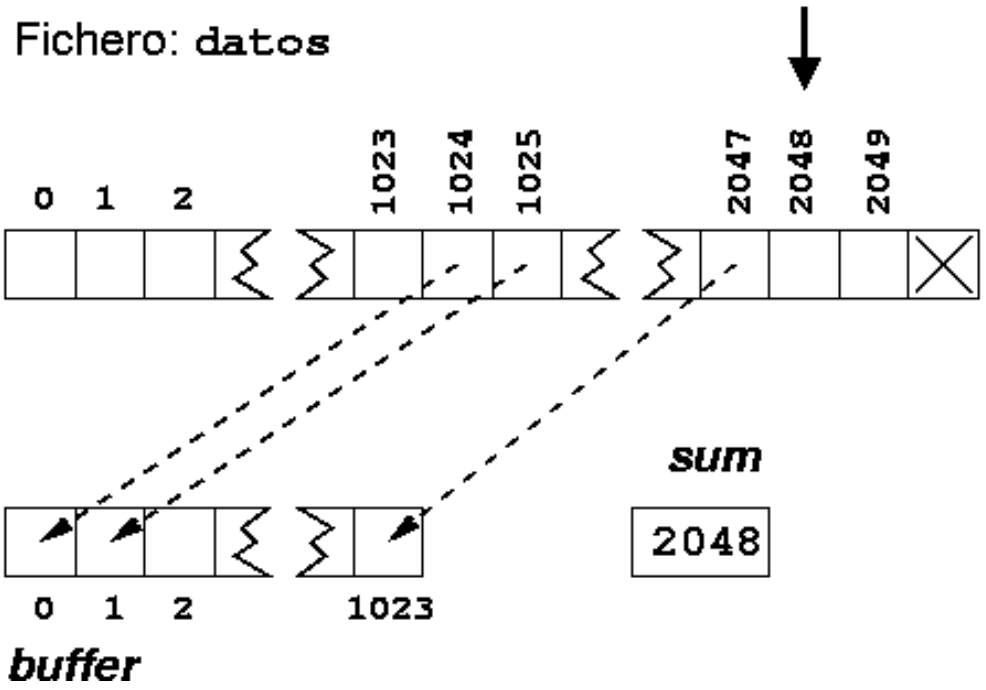
```
% tamanio2 datos

Fichero: datos      2050 bytes,     2.00 Kbytes
```



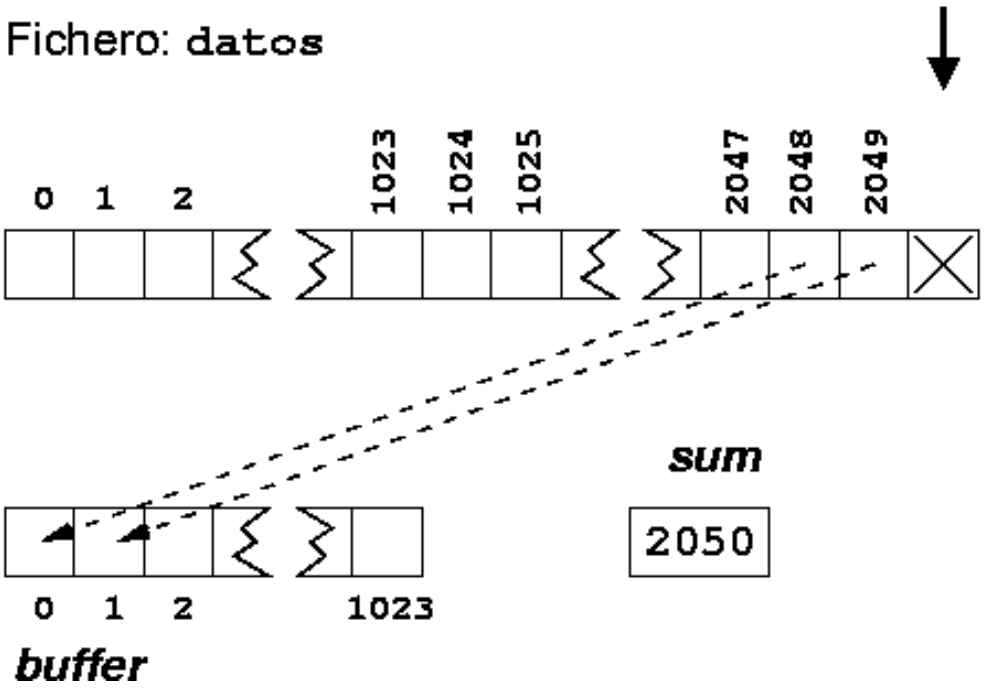
Apertura del fichero.



Lectura del primer bloque



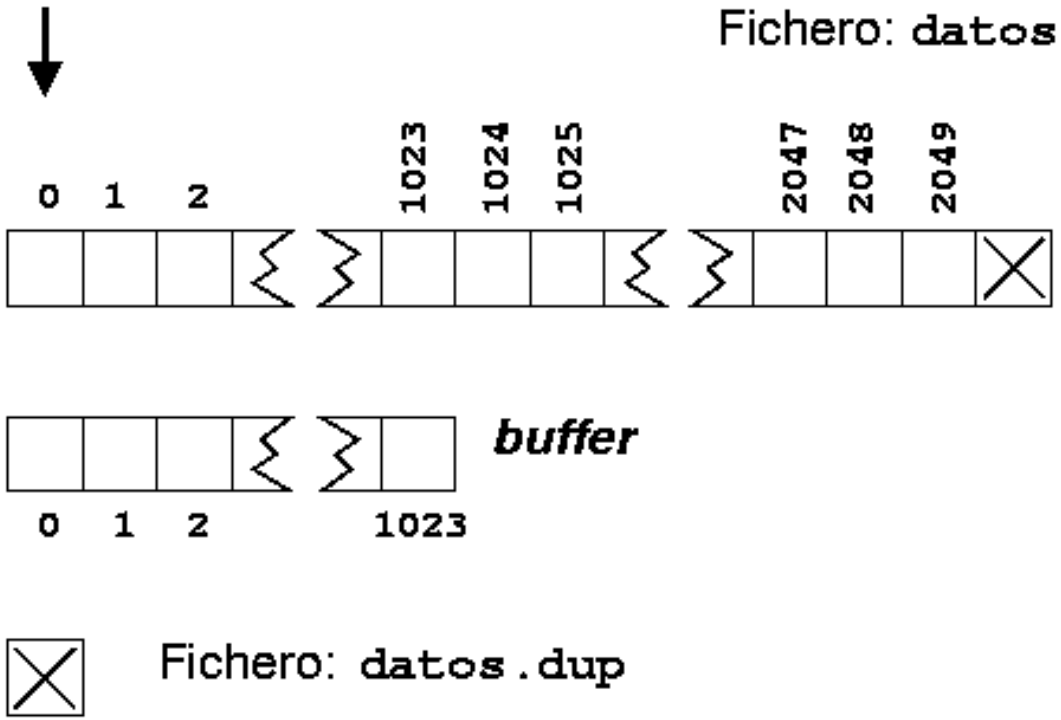
Lectura del segundo bloque



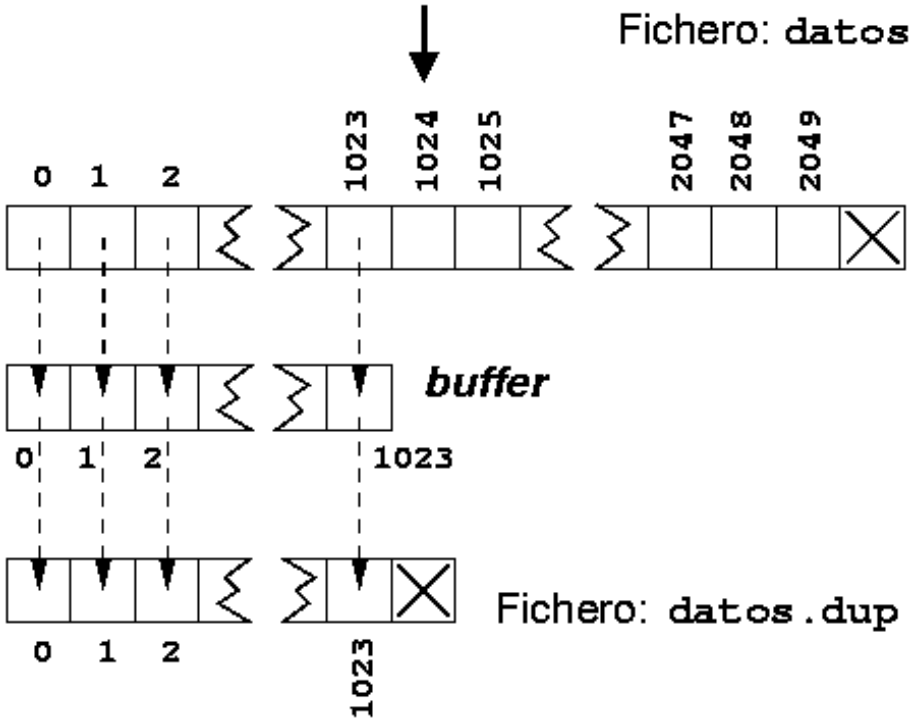
Lectura (parcial) del tercer y último bloque

```
// Fichero: copy_file2.cpp
#include <fstream>
#include <cstdlib>
using namespace std;

int main (int argc, char **argv)
{  const int TAM_BUFFER = 1024;
   unsigned char buffer [TAM_BUFFER];
   ifstream fi;
   ofstream fo;
   if (argc != 3) {
      cerr << "Error: Num. de params. erroneo\n";
      cerr << "Uso: " << argv[0]<< " <fichE> <fichS>\n";
      exit (1);
   }
   fi.open (argv[1]);
   if (!fi) {
      cerr << "Error: no puedo abrir "
```

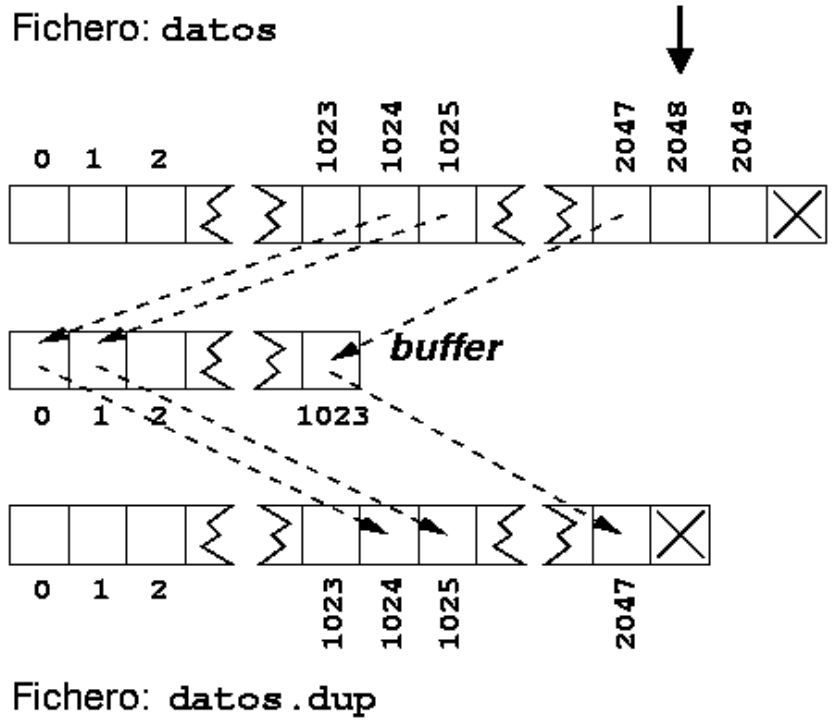
```
            << argv[1] << endl;
      exit (1);
   }
   fo.open (argv[2]);
   if (!fo) {
      cerr << "Error: no puedo crear "
            << argv[2] << endl;
      exit (1);
   }
   while (fi.read (buffer, TAM_BUFFER)) {
      fo.write (buffer, TAM_BUFFER);
   }
   fo.write(buffer, fi.gcount());
   fi.close ();
   fo.close ();
   return (0);
}
```



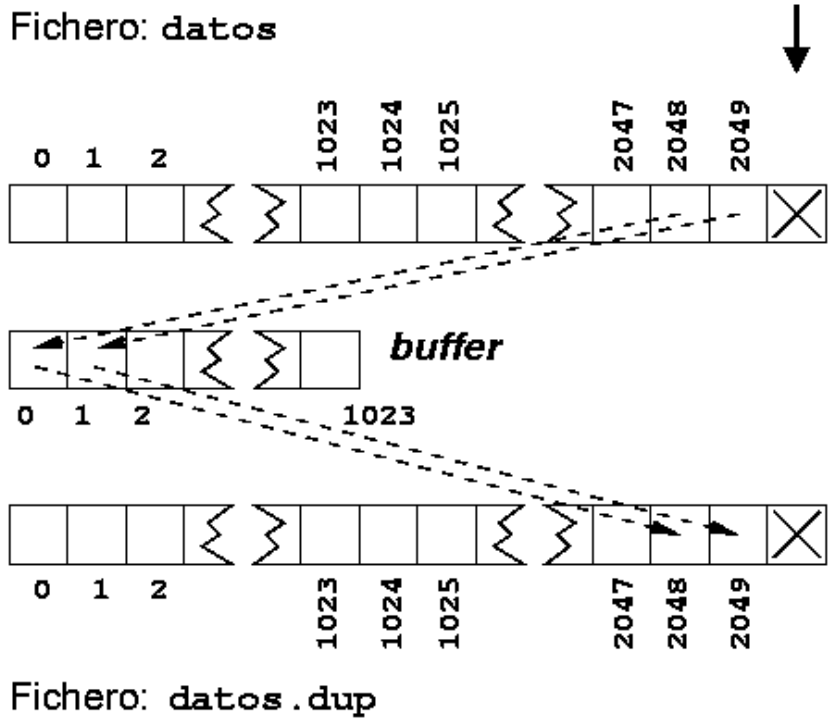
Apertura de *datos* y creación de `datos.dup`



Copia del primer bloque



Copia del segundo bloque



Copia del tercer y último bloque (no está completo)

Ejemplo

Se trata de rellenar un fichero con números aleatorios (`rellena_aleat`) y de leer su contenido (`pinta_aleat`).

```
// Fichero: rellena_aleat.cpp

#include <fstream>
#include <cstdlib>
#include <ctime>
#include <iomanip>
using namespace std;

int main (int argc, char **argv)
{
   const int MAX_LINE=8;
```

```
ofstream fo;
int num, max;
int n;

if (argc != 4) {
   cerr << "Error: Num. de params. erroneo\n";
   cerr << "Uso: " << argv[0]<< " <fichS> ";
   cerr << "<num> <max>\n";
   exit (1);
}
num = atoi(argv[2]);
if (num <= 0) {
   cerr << "Error: <num> debe ser >= 0\n";
   exit (1);
}
max = atoi(argv[3]);
if (max <= 0) {
```

```
   cerr << "Error: <max> debe ser >= 0\n";
   exit (1);
}

fo.open (argv[1]);
if (!fo) {
   cerr << "Error: no puedo crear "
        << argv[1] << endl;
   exit (1);
}
cout << "\nEl fichero " << argv[1]
     << " contendra " << num << " enteros.\n";
cout << "Sus valores estaran en el rango "
     << "0.." << (max-1) << "\n";

srand (time (NULL));
for (int i=1; i<=num; i++) {
```

```
      n = rand() % max;
      fo.write ((const char *) (&n), sizeof (int));
      cout << setw (6) << n;
      if (i%MAX_LINE==0) cout << "\n";
   }
   cout << endl;

   fo.close();
   return (0);
}
```

```
% rellena_aleat fich_aleat1 22 100
El fichero fich_aleat1 contendra 22 enteros.
Sus valores estaran en el rango 0..99
     6    99    15    64    63    88    62    99
    26    44    97    27    80    49    74    72
    92    27    13    12     0     3
```

```
// Fichero: pinta_aleat.cpp
#include <fstream>
#include <cstdlib>
#include <iomanip>
using namespace std;

bool ExisteFichero (char *nombre);
int tamanio (char *nombre);

/*****************************************/

int main (int argc, char **argv)
{
   const int MAX_LINE=8;
   ifstream fi;
   int tam, num, n;
```

```
if (argc != 2) {
   cerr << "Error: Num. de params. erroneo\n";
   cerr << "Uso: " << argv[0]<< " <fichE>\n";
   exit (1);
}

if (!ExisteFichero(argv[1])) {
   cerr << "Error: no puedo abrir "
        << argv[1] << endl;
   exit (1);
}

tam = tamanio (argv[1]);
cout << "\nFichero: " << argv[1] << " (";
cout << tam << " bytes).\n";
```

```
   num = tam / sizeof(int);
   cout << "Contiene : " << num << " enteros.\n\n";

   fi.open (argv[1]);

   for (int i=1; i<=num; i++) {
      fi.read ((char *) (&n), sizeof (int));
      cout << setw (6) << n;
      if (i%MAX_LINE==0) cout << "\n";
   }
   cout << endl;

   fi.close();
   return (0);
}
```

```
/*****************************************/

bool ExisteFichero (char *nombre)
{
   ifstream fichero;
   bool problema;

   fichero.open (nombre);
   problema = fichero.fail();
   if (!problema) fichero.close();

   return ((problema) ? false : true);
}

/*****************************************/
```

```
/*****************************************/
int tamanio (char *nombre)
{
   const int TAM_BUFFER = 1024;
   unsigned char buffer [TAM_BUFFER];
   ifstream fi;
   int sum = 0;

   fi.open (nombre);
   while (fi.read(buffer, TAM_BUFFER)) {
      sum += TAM_BUFFER;
   }
   sum += fi.gcount();
   fi.close();
   return (sum);
}
/*****************************************/
```

```
% ls -l
......
-rw-r--r--  1 pepe alumnos 88 abr 22 18:48 fich_aleat1
......

% pinta_aleat fich_aleat1

Fichero: fich_aleat1 (88 bytes).
Contiene : 22 enteros.

     6    99    15    64    63    88    62    99
    26    44    97    27    80    49    74    72
    92    27    13    12     0     3
```

5.14 Modos de apertura

- Comportamiento predeterminado de un objeto `ofstream` al abrirlo:
  - Si no existe, *crearlo*.
  - Si existe, *truncarlo* (borrar su contenido).
- Para modificarlo, dar un segundo argumento al constructor del objeto `ofstream`:

`ios::in`	Abrir para lectura.
`ios::out`	Abrir para escritura.
`ios::app`	Posición activa: final. Para añadir al final.
`ios::ate`	Posición activa: final. Puede escribirse en cualquier sitio.
`ios::trunc`	Predeterminado.
`ios::nocreate`	Si el archivo no existe, falla la apertura.
`ios::noreplace`	Si el archivo existe, falla la apertura.

Puede consultarse el resultado de la apertura con la función `fail()`.

```
// Fichero: concatena.cpp
#include <fstream>
#include <cstdlib>
#include <iomanip>
using namespace std;

void concatenar (ofstream & fo, ifstream & fi);

int main (int argc, char **argv)
{
   ifstream fi;
   ofstream fo;

   if (argc < 3) {
      cerr << "Error: Num. de params. erroneo\n";
      cerr << "Uso: "<< argv[0]<<" <fichE1> [<fichE2>";
      cerr << "...<fichEn>] <fichS>\n";
```

```
   exit (1);
}
fo.open (argv[argc-1], ios::app); // Adicion
if (!fo) {
   cerr << "Error: no puedo abrir " << argv[argc-1] << endl;
   exit (1);
}
for (int i=1; i<argc-1; i++) {
   fi.open (argv[i]);
   if (!fi) {
      cerr << "Error: no puedo abrir " << argv[i] << endl;
   }
   fo << setw(24) << setfill('-') << "-\n";
   fo << "Fichero: " << argv[i] << endl;
   fo << setw(24) << setfill('-') << "-\n";
   concatenar (fo, fi);
   fi.close();
```

```
   }
   fo.close();
   return (0);
}

/*************************************************/
void concatenar (ofstream & s, ifstream & e)
{
   const int TAM_BUFFER = 1024;
   unsigned char buffer [TAM_BUFFER];

   while (e.read (buffer, TAM_BUFFER)) {
     s.write (buffer, TAM_BUFFER);
   }
   s.write(buffer, e.gcount());
}
/*************************************************/
```

```
% concatena letra_A letra_B resultado
% more resultado
----------------------
Fichero: letra_A
----------------------
A
----------------------
Fichero: letra_B
----------------------
B
% concatena fichero_eco resultado
% more resultado
----------------------
Fichero: letra_A
----------------------
A
----------------------
```

```
Fichero: letra_B
----------------------
B
----------------------
Fichero: fichero_eco
----------------------
Esto es una
prueba de redireccion
```

5.15 Funciones de posicionamiento

```
istream & seekg (int desp);
istream & seekg (int desp, int origen);
ostream & seekp (int desp);
ostream & seekp (int desp, int origen);
```

Establece la posición actual del fichero sobre el que se aplica.

1. Con un parámetro (`desp`) se efectúa un posicionamiento absoluto (`desp` bytes) desde el principio del fichero.
2. Con dos parámetros, el posicionamiento es relativo respecto al `origen`, que puede ser:

`ios::beg` : inicio del fichero,
`ios::cur` : posición actual,
`ios::end` : fín del fichero.

```
int tellg ();
int tellp ();
```

Devuelve la posición actual en un fichero de entrada (`tellg()`) o salida (`tellp()`).

```
// Fichero: reves.cpp

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

int main (int argc, char **argv)
{
   ifstream fi;
   char c;

   if (argc != 2) {
      cerr << "Error: Num. de params. erroneo\n";
      cerr << "Uso: " << argv[0]<< " <fichE>\n";
      exit (1);
   }
```

```
fi.open (argv[1]);
if (!fi) {
   cerr << "Error: no puedo abrir "
        << argv[1] << endl;
   exit (1);
}

fi.seekg ( 0, ios::end); // Colocar la final
fi.seekg (-1, ios::cur); // Retroceder 1

while (fi.tellg() > 0) {
   c = fi.get();
   cout.put(c);
   fi.seekg (-2, ios::cur); // Retroceder 2
}
c = fi.get(); // Leer el primero
```

```
   cout.put(c);  // Escribir el primero

   fi.close ();

   return (0);
}

% cat fichero_eco
Esto es una
prueba de redireccion

% reves fichero_eco
noiccerider ed abeurp
anu se otsE%
```

```
// Fichero: tamanio3.cpp

#include <fstream>
#include <cstdlib>
#include <string>
#include <iomanip>
using namespace std;

int tam_file (char *nombre);

int main (int argc, char **argv)
{
   int tam;

   if (argc != 2) {
      cerr << "Error: Num. de params. erroneo\n";
      cerr << "Uso: " << argv[0]<< " <fich>\n";
```

```
      exit (1);
   }
   tam = tam_file (argv[1]);

   cout.setf (ios::fixed);
   cout << "\nFichero: " << argv[1] << setw(10) << tam << " bytes, ";
   cout << setprecision(2) << setw(8) << tam/1024.0 << " Kbytes\n\n";

   return (0);
}
```

```
/*************************************************/
int tam_file (char *nombre)
{
   ifstream fi;
   int tam;

   fi.open (nombre);
   if (!fi) {
      cerr << "Error: no puedo abrir " << nombre << endl;
      exit (1);
   }
   fi.seekg ( 0, ios::end); // Colocar la final
   tam = fi.tellg();
   fi.close ();
   return (tam);
}
/*************************************************/
```

Esta función se puede sobrecargar:

```
int tam_file (string & nombre);
```

y se llama:

```
string nombre;
......
nombre = argv[1];
tam = tam_file (nombre);
```

```
/************************************************/

int tam_file (string & nombre)
{  ifstream fi;
   int tam;

   fi.open (nombre.c_str());
   if (!fi) {
      cerr << "Error: no puedo abrir " << nombre << endl;
      exit (1);
   }
   fi.seekg ( 0, ios::end); // Colocar la final
   tam = fi.tellg();
   fi.close ();
   return (tam);
}
/************************************************/
```