

# ***Programación Orientada a Objetos***

## **Unidad 6: Flujos de Entrada/Salida en C++**

*Teoría 08 - 20/10/2011 - Pablo Novara*

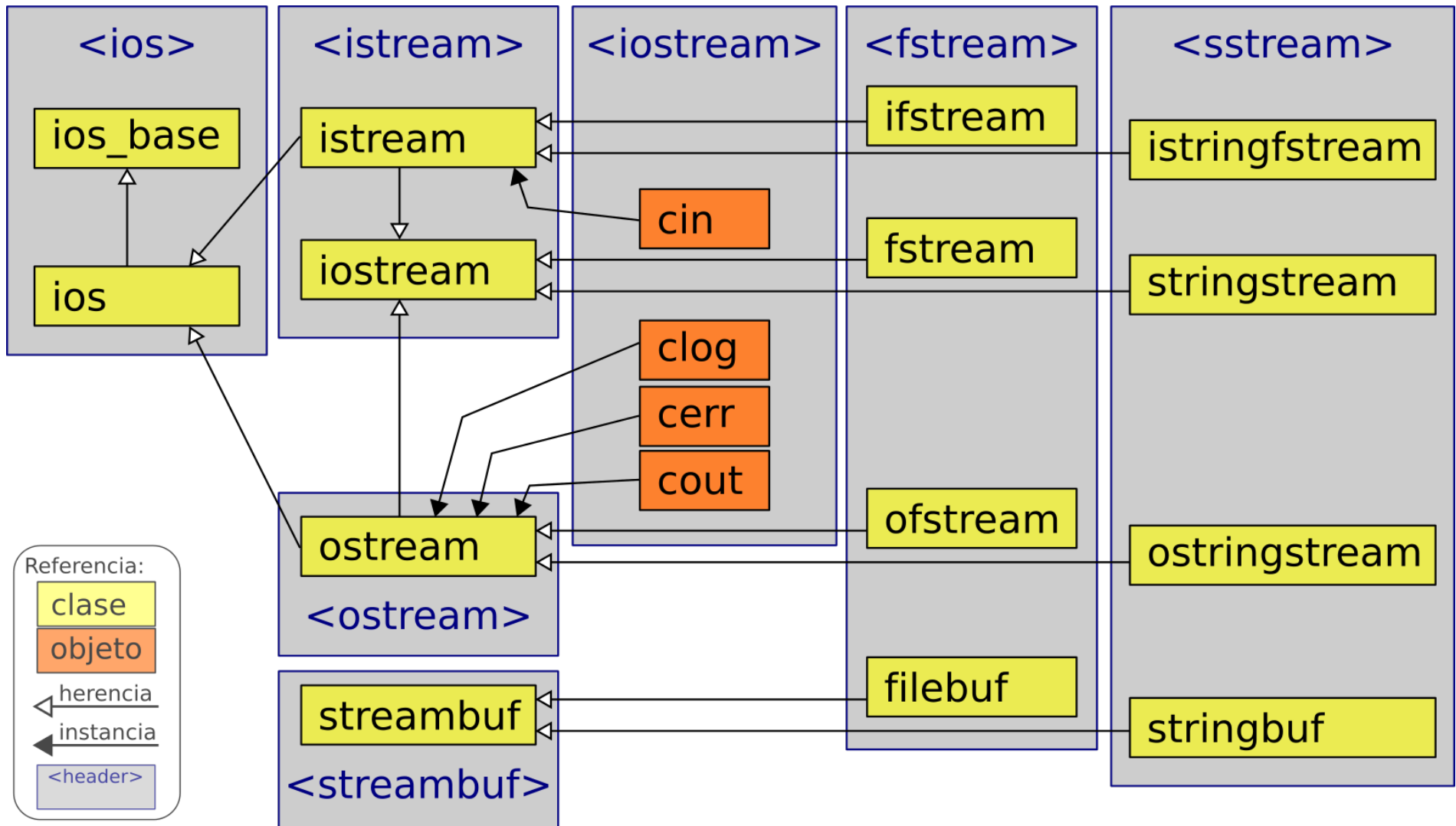
## ¿Qué son los streams?

El stream (flujo) es el tipo de dato básico para realizar operaciones de entrada/salida en C++.

- Algunas instancias de streams ya conocidas son:
  - `cin`: entrada de texto desde la consola (teclado)
  - `cout`: salida de texto hacia la consola
- C++ hereda también los mecanismos de I/O de C, pero los streams proveen casi toda su funcionalidad con una interfaz mejorada.

## Streams en C++

- C++ presenta una jerarquía de objetos relacionados a flujos de entrada y salida:



## Streams y Archivos

– Hay tres usos básicos para streams en C++:

- Leer del teclado/escribir en pantalla:

instancias `cin/cout/cerr/clog`

- Leer/escribir en strings:

clase `stringstream`

- Leer escribir **en archivos**:

clases `fstream/ifstream/ofstream`

## ¿Qué es un archivo?

- En informática (según consta en Wikipedia):  
es un conjunto de bits almacenado en un dispositivo
- En los sistemas de archivos modernos, se almacena y se lee por bytes (8 bits), y se identifican con un nombre y una ubicación (unidad y carpetas/directorios).
- Además, se le pueden otorgar ciertos “permisos” o “atributos” (escritura/lectura/ejecución/oculto,etc) según el sistema de archivos.

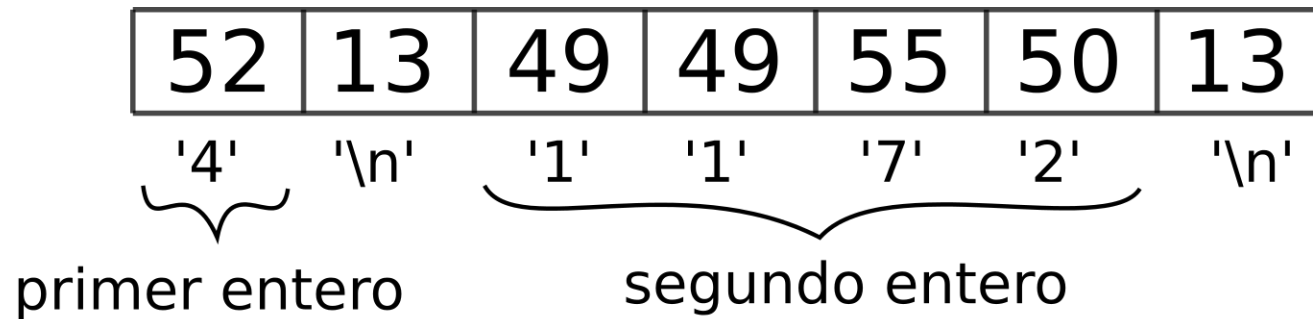
## Archivos de Texto vs Archivos Binarios

- Las categorías “**de texto**” y “**binario**” hacen referencia a **la forma de “codificar” la información** que se guarda:
  - En un archivo de texto, cada byte representa un carácter según el código ASCII, y todo el archivo se interpreta como texto.
  - En un archivo binario, los datos se exactamente guardan como se guardan en memoria.
- **Para el sistema de archivos, no hay ninguna diferencia** entre un tipo de archivo y el otro.

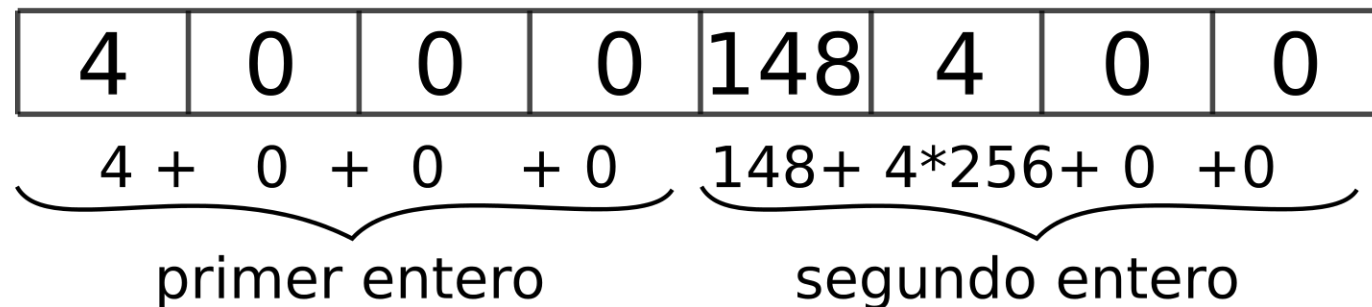
## Archivos de Texto vs Archivos Binarios

– Los enteros 4 y 1172:

- Archivo de Texto:



- Archivo Binario:



## Archivos de Texto vs Archivos Binarios

### – Archivos de texto:

- Se pueden leer/modificar fácilmente con cualquier editor de texto (por ejemplo notepad).
- Se pueden escribir y leer sin problemas y de igual forma desde cualquier sistema operativo, ya que el código ascii es siempre el mismo.
- Si se guardan muchos datos de igual tipo (nombres, notas, telefonos, etc), en general, no todos ocupan la misma cantidad de bytes.

– En consecuencia, su acceso es **secuencial**.

- Los datos que no son cadenas de texto, se convierten al escribir/leer.

## Archivos de Texto vs Archivos Binarios

### – Archivos binarios:

- Los datos se guardan directamente sin conversión.
- No se pueden leer ni modificar fácilmente, ya que hay que conocer su organización.
  - Usualmente solo el programa que los creó puede editarlos correctamente.
  - Aun conociendo los tipos de datos guardados, su verdadera codificación puede variar de un sistema operativo a otro.
- Si se guardan muchos datos de igual tipo, en general, todos ocupan la misma cantidad de bytes.
  - En consecuencia, su acceso es **aleatorio**.

## Archivos de Texto vs Archivos Binarios

### – Ventajas y Desventajas:

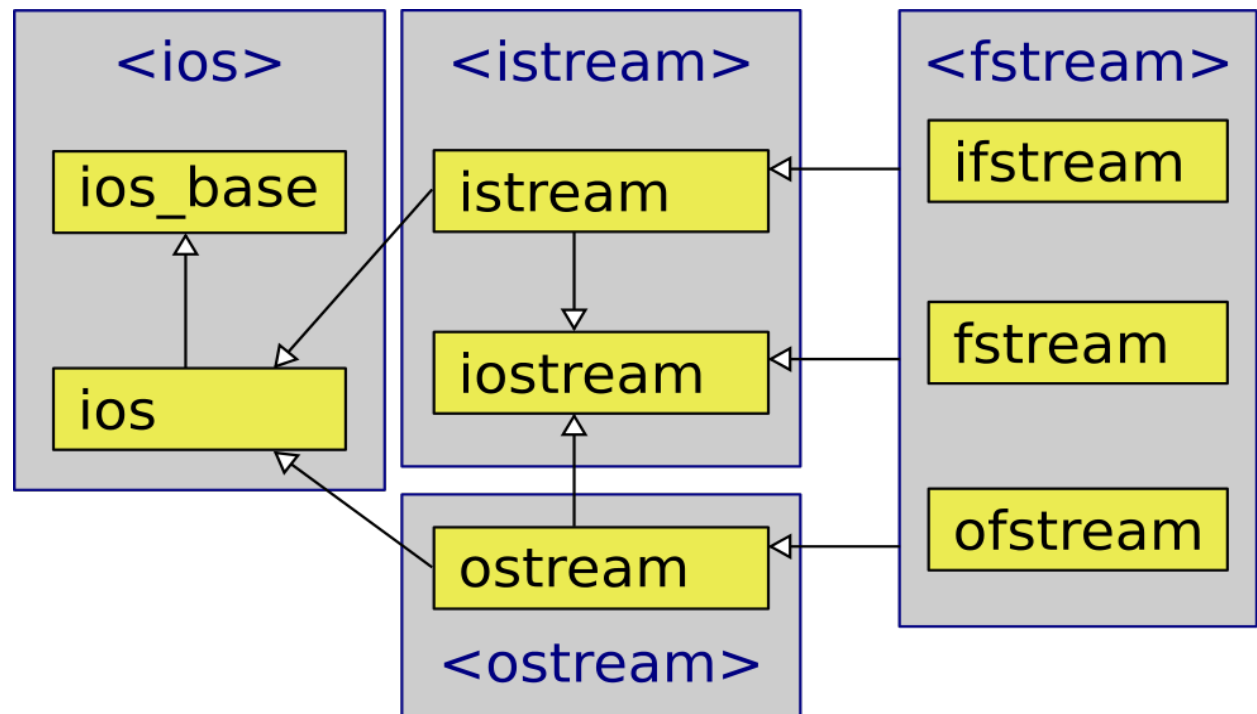
- Los archivos de texto se pueden pasar de un programa a otro sin problemas:
  - Un archivo de texto se puede ver/editar fácilmente con cualquier editor de texto.
- Operar con archivos binarios es más rápido porque no implica conversión.
- Los archivos binarios permiten acceso aleatorio:
  - esto posibilita leer y escribir en un mismo archivo con un mismo objeto.

## Archivos de Texto vs Archivos Binarios

- ¿Como operar sobre los datos de un archivo?
  - Opción 1: cargarlo en memoria al comienzo, operar en memoria, guardarlo completo antes de finalizar
    - Bases de datos pequeñas
    - Archivos sin formato
    - Texto o datos de longitud variable
  - Opción 2: operar directamente sobre el archivo
    - Bases de datos grandes (ej: históricos, logs)
    - Si hay que modificar, solo con archivos binarios
    - Hay que borrar sin borrar

## Archivos en C++

- Para acceder a archivos se utilizan las clases:
  - **ifstream**: para lectura
  - **ofstream**: para escritura
  - **fstream**: para lectura y/o escritura



## Archivos en C++

- Antes de utilizar un archivo, se debe “abrir”. Esto significa, asociar el **objeto** con una **dirección/ruta** en el sistema de archivos:

```
ifstream archi("notas.txt");  
  
if (!archi.is_open()) {  
    cout<<"No se pudo abrir el archivo.\n";  
    return 1;  
}
```

Luego de utilizarlo, se lo debe cerrar:

```
archi.close();
```

## Archivos en C++

- Dos formas de abrir un archivo:
  - Mediante el constructor de fstream:
    - `ifstream archi(“notas.txt”);`
  - Mediante el método open:
    - `ifstream archi;`  
`archi.open(“notas.txt”);`
- En ambos casos, se puede añadir un segundo argumento con **banderas** que indican “cómo” abrir el archivo:
  - `fstream archi( “notas.txt”, ios::in | ios::binary )`

## Archivos en C++

- Las posibles banderas para `fstream::open` son:
  - `ios::in` indica que se abre para lectura, o que no se debe eliminar el contenido
  - `ios::trunc` indica que se debe eliminar todo el contenido del archivo
  - `ios::out` indica que se abre para escritura
  - `ios::app` indica que se escribe siempre al final
  - `ios::ate` indica que el “cursor” se ubica al final al abrir el archivo
  - `ios::binary` indica que se accederá en modo binario

## Archivos en C++

- Las clases de `fstream` tienen métodos para consultar el estado del stream, retornando `true` o `false`:
  - `is_open()` indica si se abrió correctamente el archivo
  - `fail()` indica si hubo un error cualquiera
  - `bad()` indica si hubo un error irreparable
  - `eof()` indica si se intentó leer más allá del final del archivo
  - `good()` retorna `true` si no tiene errores y está listo para la lectura/escritura
- El método `clear()` permite resetear el estado.

## Archivos de Texto en C++

– Para leer/escribir en archivos de texto se utilizan los mismos métodos, funciones, y operadores que para leer/escribir en consola:

– Se escribe con << y se pueden utilizar manipuladores:

```
ofstream archi("notas.txt");  
archi<<nombre<<endl;  
archi<<fixed<<set_precision(2)<<prom<<endl;
```

– Se lee con >> y/o getline:

```
ifstream archi("notas.txt");  
getline(archi, nombre);  
archi>>nota;
```

## Archivos de Texto en C++

- Para modificar un archivo de texto, hay que reescribirlo completamente.

Ejemplo 1) escriba un programa para leer una lista de nombres y notas de 3 parciales de un curso de POO de un archivo de texto como el siguiente:

```
Carmack, Juan  
10 9 10  
Gates, Guillermo  
6 5 7  
Stallman, Ricardo  
7 10 8  
...
```

y reemplace las notas por el promedio.

## Archivos Binarios en C++

- Para trabajar con un archivo en modo binario, se debe añadir la bandera `ios::binary` al modo de apertura:

```
ifstream archi("datos.dat",ios::binary);
```

- Los datos se leen/escriben con los métodos `read` y `write`.
  - El primer argumento es un **puntero de tipo char** que apunta al dato que se quiere leer/escribir:
  - El segundo argumento es la **cantidad de bytes** que se leen/escriben.

```
int x;  
archi.write( (char*)&x , sizeof(x) );
```

## Archivos Binarios en C++

Ejemplos:

2) Escriba un programa que permita ingresar pares de datos compuestos por un double y un int y los guarde en un archivo cuyo nombre ingresa el usuario.

3) Escriba un programa que solicite el nombre de un archivo generado con el programa del ejemplo 2 y muestre su contenido.

## Archivos Binarios en C++

- No es correcto guardar instancias de clases que utilizan punteros en un archivo binario.
  - Se guardan los punteros, pero no los datos apuntados!

```
string s1="Hola Mundo!";  
ofstream a1("archi.dat",ios::binary);  
a1.write((char*)&s,sizeof(s));  
a1.close();
```

“Anda”, pero  
está ¡MAL!

```
string s2;  
ifstream a2("archi.dat",ios::binary);  
a2.read((char*)&s2,sizeof(s2));  
cout<<s2; // muestra “Hola Mundo!”
```

## Archivos Binarios en C++

- Si no se especifica otra cosa, los datos se leen o escriben uno a continuación de otro.
- Dado que en modo binario se dispone de acceso aleatorio, hay métodos para modificar o consultar la posición donde se leerá o escribirá la próxima vez:
  - `tellg()/tellp()` devuelve la posición en que se encuentra el puntero.
  - `seekg()/seekp()` permiten establecer la posición del puntero
- Hay un único puntero para leer y escribir.

## Archivos Binarios en C++

– Ejemplos:

4) Escriba un programa para generar 20 enteros aleatorios y guardarlos en un archivo binario.

5) Escriba un programa para buscar el mayor entero del archivo generado en el ejemplo anterior y reemplazarlo por -1.