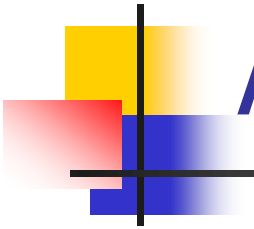
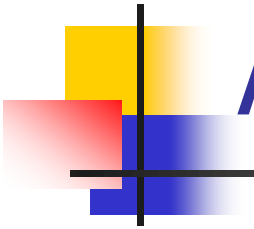


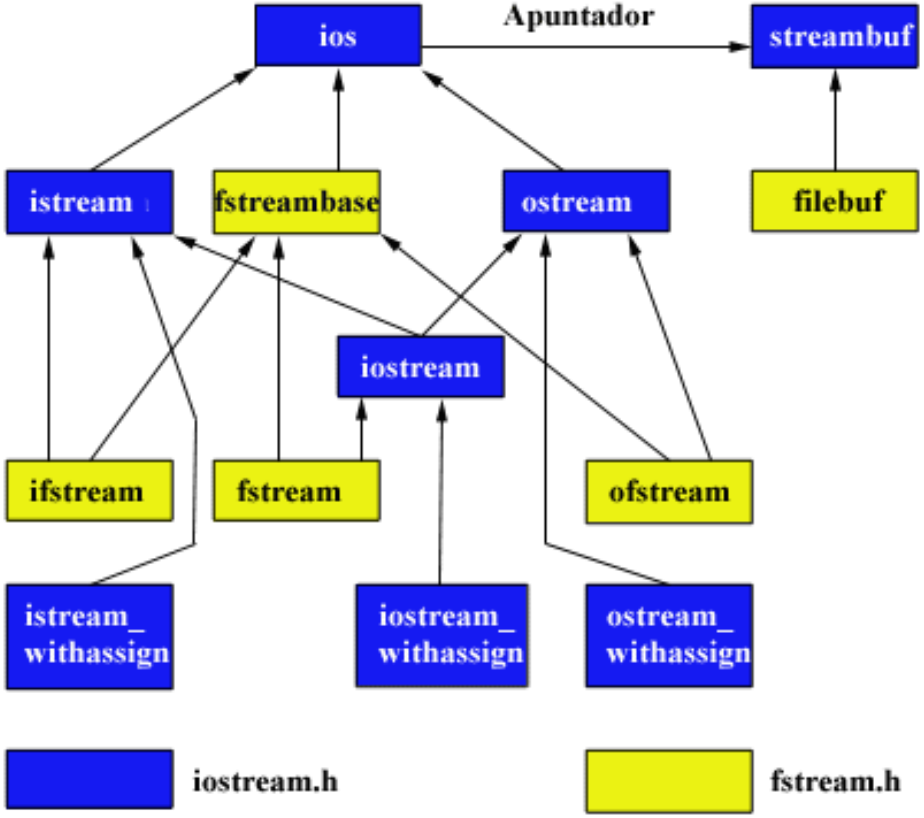
Archivos en C++

Archivos y streams



- C++ ve a cada archivo simplemente como una secuencia de bytes
- Un archivo finaliza ya se con una marca de *end-of-file* (eof) o en un byte determinado (tamaño del archivo) en una estructura de datos administrada por el sistema operativo (como un archivo en un directorio)
- Cuando un archivo se abra, se instancia un objeto archivo del tipo adecuado y un stream (también llamado nombre de archivo lógico) es asociado con este objeto.
- Para realizar estas acciones en C++ deben incluirse los archivos **<iostream> y <fstream>**

Jerarquía de clases stream



Jerarquía de clases stream

Apertura de archivos

```
ofstream archivo;
archivo.open ("ejemplo.bin", ios::out | ios::app | ios::binary);
```

ios::in	Abrir archivo para lectura
ios::out	Abrir archivo para escritura
ios::ate	Posición inicial: fin del archivo
ios::app	Cada salida se agrega al final del archivo
ios::trunc	Si el archivo ya existe se borra su contenido
ios::binary	Modo binario

```
ofstream archivo ("ejemplo.bin", ios::out | ios::app | ios::binary);
```

Funciones de archivos

- Cerrar un archivo: Se encarga de vaciar los buffers que usa el archivo y cerrar el archivo.

  Archivo.close();

- Función eof(): para conocer si se ha llegado al final del archivo al leerlo.

Opciones de estado

- Permiten conocer el estado de los flujos de datos

- **bad() :**Retorna **true** si ocurre una falla en las operaciones de lectura o escritura. Por ejemplo en caso de que tratemos de escribir en un archivo que no está abierto para escritura.
- **fail()** Retorna **true** en los mismos casos que **bad()** y además en caso de que ocurra un error de formato, como tratar de leer un número entero y obtener una letra.
- **eof()** Retorna **true** si el archivo abierto ha llegado a su fin.
- **good()** Es el más genérico: Retorna **false** en los mismos casos en los que al llamar a las funciones previas hubiéramos obtenido **true**.
- Ejemplos errfile01,errfile02

Archivos de texto

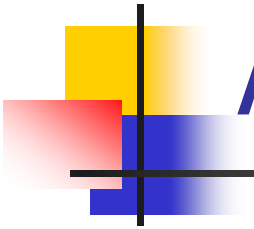
- Se organiza la secuencia de bits de manera de formar caracteres.
- De esta manera funciona cin y cout, por lo que podemos utilizar los operadores de redirección << y >>.

```
// escribiendo en un archivo de texto
#include <fstream.h>
int main () {
ofstream ejemplo("ejemplo.txt");
if (ejemplo.is_open())  {
  ejemplo << "Esto es una línea.\n";
  ejemplo << "Esto es otra línea.\n";
  ejemplo.close();
 }
return 0;}
```

Lectura de archivos de texto

```
// leyendo un archivo de texto
#include <iostream>
#include <fstream.h>

int main () {
char buffer[256];
ifstream ejemplo ("ejemplo.txt");
if (! ejemplo.is_open())   {
  cout << "Error al abrir el archivo";
  exit (1); }
while (! ejemplo.eof() )   {
   ejemplo.getline (buffer,100);
  cout << buffer << endl;   }
return 0;}
```

```
Esto es una línea.
Esto es otra línea.
```



Archivos binarios

- Limitación de los archivos de texto de los datos a almacenar.
- Problemas en la lectura de caracteres como eof.
- Tamaño de los elementos almacenados.
- Procesamiento implícito o explícito de lo que se lee.

Archivos binarios (ejemplo)

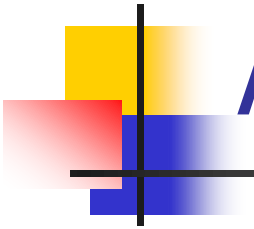
```
// readbin.cpp leyendo archivos binarios
#include <iostream>
#include <fstream.h>

const char * nombre_archivo = "ejemplo.dat";
int main () {
  char * buffer;
  long tamanio;
  ifstream file (nombre_archivo,ios::in|ios::binary|ios::ate);
  tamanio = file.tellg();
  buffer = new char [tamanio];
  file.read ((char*) & buffer, tamanio);
  file.close();
  cout << "El archivo completo está en el buffer";
delete[] buffer;
return 0;}
```

El archivo completo está en el buffer

Archivo binario

```
#include <fstream.h>
#include <stdlib.h>
#include <string.h>
struct tipoRegistro {
        char nombre[32];
        int edad;
float altura; };
int main() {
tipoRegistro pepe; tipoRegistro pepe2;
ofstream fsalida("prueba.dat", ios::out | ios::binary);
strcpy(pepe.nombre, "Jose Luis");
pepe.edad = 32;
pepe.altura = 1.78;
fsalida.write((char *)&pepe, sizeof(tipoRegistro));
fsalida.close();
ifstream fentrada("prueba.dat", ios::in | ios::binary);
fentrada.read((char *)&pepe2, sizeof(tipoRegistro));
cout << pepe2.nombre << endl;
cout << pepe2.edad << endl;
cout << pepe2.altura <<endl;
fentrada.close();
system("PAUSE"); return 0; }
```

Archivos binarios



- Se pueden leer y escribir simultáneamente. Ej lecescsec.cpp

Archivos de entrada y salida

```
//lecescsec.cpp
// lectura-escritura en archivos binarios
#include <fstream>
#include <iostream>
int main(void)
{ fstream LeerEscribir("valores.dat",ios::binary | ios::in | ios::out);
  float f=8.1; int i=4;
  LeerEscribir.write( &f,sizeof(f) );
  LeerEscribir.write( &i,sizeof(i) );
  LeerEscribir.seekg( -sizeof(i) );
  LeerEscribir.read ( &i,sizeof(i) );
  cout << i << endl;
  LeerEscribir.seekp(sizeof(f),ios::beg);
  f=3.56;
  LeerEscribir.write( &f,sizeof(f) );
  LeerEscribir.seekp( 0,ios::beg );
  LeerEscribir.read ( &f,sizeof(f) );
  LeerEscribir.read ( &i,sizeof(i) );
  cout << f << " " << i << endl;
  LeerEscribir.close();
return (0);
}
```

Punteros de flujo get y put

- Funciones para obtener posición

  tellg() tellp()

- Funciones para posicionarse

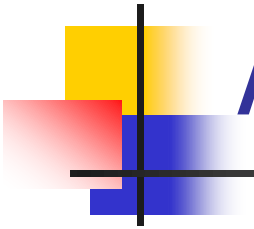
  seekg() seekp()

seekg (off_type *desplazamiento*, **seekdir** *dirección* **);**

seekp (off_type *desplazamiento*, **seekdir** *dirección* **);**

ios::beg	Desplazamiento especificado desde el principio del flujo
ios::cur	Desplazamiento especificado desde la posición actual del puntero
ios::end	Desplazamiento especificado desde el final del flujo

Archivos de acceso aleatorio



- Permiten acceso secuencial y aleatorio.
- Es posible cambiar sus valores sin tener que escribir nuevamente el archivo.
- Primer implementación:
- Todos los registros del archivo son del mismo tipo y tamaño.

Acceso aleatorio

```
#include <fstream.h>
#include <stdlib.h>
int main() {
int i; char cad[20];
char mes[][20] = {"Enero", "Febrero", "Marzo", "Abril", "Mayo",
"Junio", "Julio", "Agosto", "Septiembre", "Octubre",
"Noviembre", "Diciembre"};
// Crear fichero con los nombres de los meses:
ofstream fsalida("meses.dat", ios::out | ios::binary);
cout << "Crear archivo de nombres de meses:" << endl;
for(i = 0; i < 12; i++) fsalida.write(mes[i], 20);
fsalida.close();
ifstream fentrada("meses.dat", ios::in | ios::binary);
// Acceso secuencial:
cout << "\nAcceso secuencial:" << endl;
fentrada.read(cad, 20);
do {
        cout << cad << endl; fentrada.read(cad, 20); }
while(!fentrada.eof());
fentrada.clear();
// Acceso aleatorio:
cout << "\nAcceso aleatorio:" << endl;
for(i = 11; i >= 0; i--) {
    fentrada.seekg(20*i, ios::beg);
    fentrada.read(cad, 20);
cout << cad << endl; } // Calcular el número de
elementos almacenados en un fichero:
fentrada.seekg(0, ios::end); // ir al final del fichero

pos = fentrada.tellg();
// leer la posición actual
// El número de registros es el tamaño en bytes dividido
entre el
// tamaño del registro:
cout << "\nNúmero de registros: " << pos/20 << endl;
fentrada.close();
system("PAUSE");
return 0; }
```