

**A**lgoritmos y

**E**structuras

**D**e

**D**atos

# **Datos**

**en memoria externa:**

- **ARCHIVOS (Ficheros)**

# Introducción

- Un **archivo** es una colección de datos almacenados juntos bajo un nombre común.
  - ✓ Los programas en C++ son ejemplos de archivos.
- Los archivos **almacenan información de manera permanente** en dispositivos de almacenamiento de memoria secundaria, tales como dispositivos magnéticos (discos duros, cintas), discos ópticos (CDROM, DVD), memorias permanentes de estado sólido (memorias flash USB), etc.
  - ✓ La información puede ser tanto programas (software) como datos que serán utilizados por los programas.
- Los archivos de datos pueden ser creados, leídos y actualizados por programas en C++.

# Archivos (ficheros)

- Para tratar con un fichero se utilizan, aparte de otras operaciones, unas operaciones básicas que son: **abrir, cerrar, escribir y leer**.
- Los ficheros pueden ser **de lectura**, en cuyo caso diremos que el tipo de acceso del fichero es de entrada, o bien **de escritura**, con lo cual tendremos un fichero de salida.
- Los ficheros de salida pueden **crearse** (o sobreescribirse, si ya existen) o bien **actualizarse**.
  - ✓ También se dispone de ficheros de **entrada / salida**.
- El tipo de acceso determinará las **operaciones disponibles** para el fichero.

La biblioteca que permite la manipulación de ficheros recibe el nombre de **fstream**.

Esta biblioteca nos provee de objetos que serán de entrada (**ifstream**) o de salida (**ofstream**).

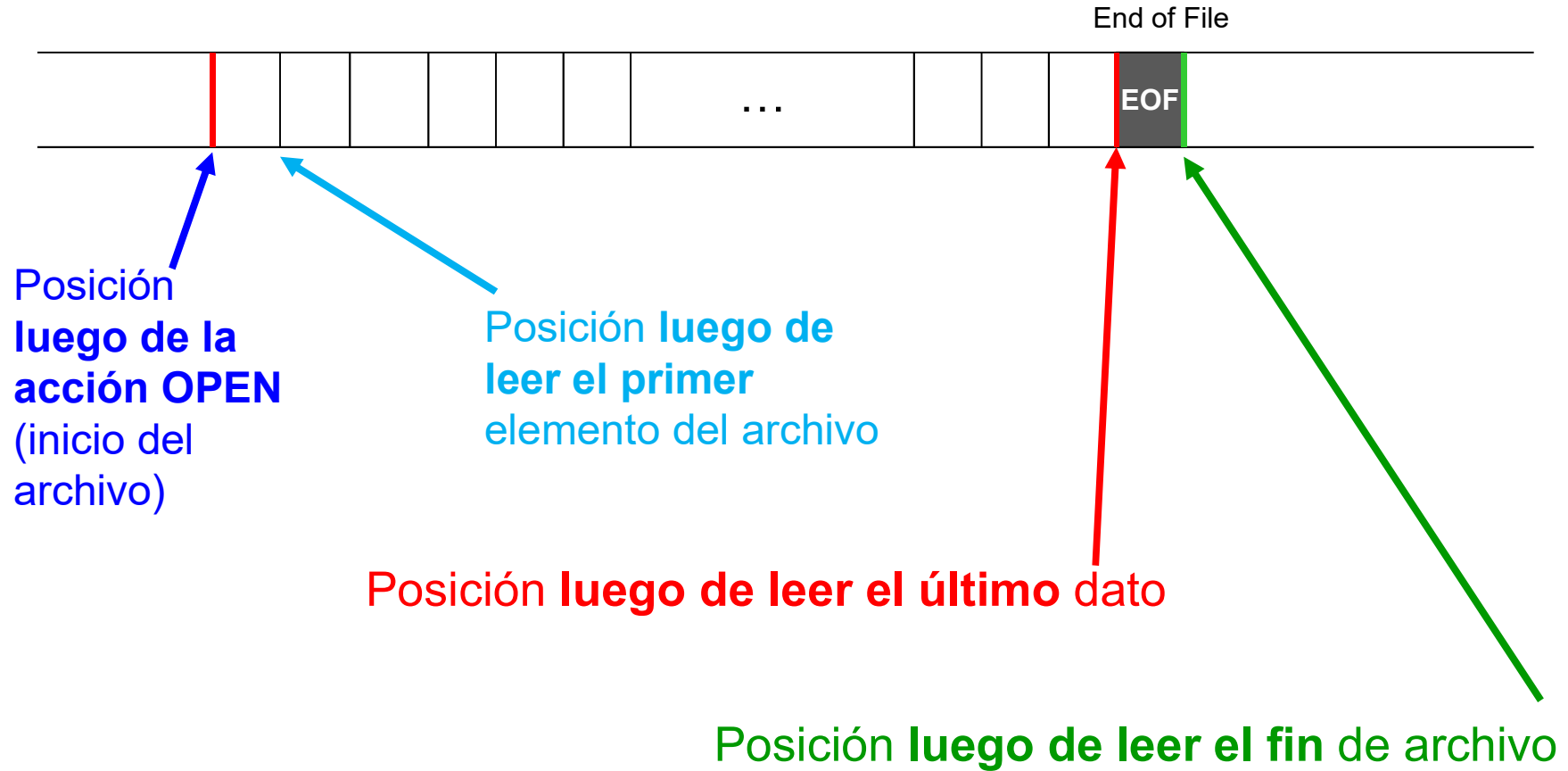
Todos los ficheros disponen de operaciones para abrirlos, cerrarlos y preguntar si estamos al final.

### **Información relativa a un fichero**

Los objetos del tipo fichero son objetos lógicos (variables de nuestro programa) que representan un fichero físico.

Para poder identificar de forma unívoca el fichero físico que manejan, se necesita un nombre de fichero.

# Información en archivos

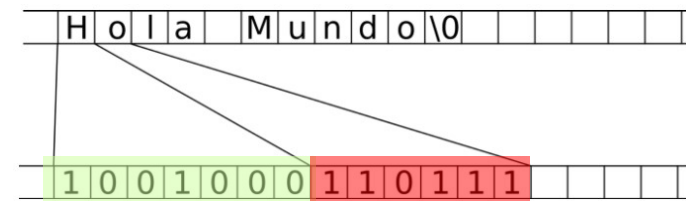


# Tipos de Archivos

Existen dos tipos de archivos, que **se distinguen de acuerdo al formato en que almacenan la información**:

- **De texto:** La información se almacena como una secuencia de caracteres; cada carácter individual se almacena utilizando una codificación estándar (usualmente basada en la codificación ASCII). Al tratarse de un formato estandarizado, otros programas diferentes de aquel que creó el fichero podrán entender y procesar su contenido.

Los archivos de texto al fin y al cabo son también archivos binarios, pero los tratamos como un tipo de archivo distinto, porque los podemos ver y editar desde un editor de texto.



- **Binarios:** La información se almacena con el mismo formato y codificación utilizada por el compilador C++ para sus datos primarios:
  - Los *números* aparecen en su forma binaria verdadera
  - Las *cadenas* conservan su forma ASCII

La ventaja de los archivos binarios es su *compactibilidad* debido a que se usa menos espacio para almacenar más números usando su código binario que como valores de carácter individuales.

Si queremos guardar el número 123456 en un archivo de texto, cada dígito ocupará 8 bytes. Usando un formato binario, el número 123456 ocupará 4 bytes (lo que ocupa un int en C++).

# Clasificación según tipo de acceso

Por la manera de acceder a los archivos, los podemos clasificar en función de:

## La dirección del flujo de los datos:

- **De entrada:** Son aquellos cuyos datos se leen por parte del programa.
- **De salida:** Aquellos ficheros que el programa escribe.
- **De entrada/salida:** Ficheros que se pueden leer y escribir.

## Dependiendo de cómo se accede a los datos en sí:

- **Secuencial:** El orden de acceso a los datos está determinado, primero se accede al primer elemento y luego se puede ir accediendo a los siguientes, de uno en uno.
- **Directo / Aleatorio:** Se puede acceder de forma directa a un elemento concreto del fichero. En este caso, el acceso es similar al de los arreglos.



## Clasificación según la estructura de la información

- En los **ficheros de texto** la información se almacena como una secuencia de caracteres y cada carácter se almacena utilizando una **codificación estándar** (usualmente basada en la codificación ASCII) y al tratarse de un formato estandarizado, otros programas diferentes de aquel que creó el fichero podrán entender y procesar su contenido

# Flujo de E/S asociados a ficheros

- Se dispone de un fichero de texto: “**fechas.txt**”
- Almacenado en memoria: [home/alumno/documentos/fechas.txt](#)
- Que contiene información según el siguiente formato, donde cada línea se encuentra terminada por un carácter (no visible) terminador de fin de línea

```
Juan López 12 3 1992
Lola Martínez 23 7 1987
Pepe Jiménez 17 8 1996
```

- Aunque los datos almacenados en memoria se encuentran en formato binario, **son convertidos a su representación textual antes de ser escritos en el fichero.**
- Similarmente, cuando leemos del fichero de texto para almacenar la información en memoria **se produce una conversión de formato de texto a fomato binario.**

# Entrada y salida de información

- Un programa codificado en C++ realiza la entrada y salida de información a través de **flujos de entrada y salida** respectivamente.
- La entrada y salida de datos a través de los **flujos estándares** de entrada y salida (**cin y cout**, respectivamente), usualmente conectados con el teclado y la pantalla de la consola.
- **Todo lo visto anteriormente respecto a la entrada y salida** básica con los flujos estándares, o la entrada y salida de cadenas de caracteres, también **es aplicable** a los flujos de entrada y salida vinculados a **ficheros**.



## Ejemplo 1. Un archivo de texto de entrada se comporta como cin

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

```
int main() {
    int a; char z; double c;
    cout << "Entrada desde teclado: " << endl;
    cin >> z >> a >> c;
    cout << "Salida: " << endl;
    cout << z << " " << a << " " << c << endl << endl;

    ifstream f;
    f.open("archi.txt");

    cout << "Entrada desde archivo: " << endl;
    f >> z >> a >> c;
    cout << "Salida: " << endl;
    cout << z << " " << a << " " << c ;
    f.close();
    return 0;
}
```



archi: Bloc de notas

Archivo Edición Formato Ver Ayuda

h 23 45.7



Entrada desde teclado:

s 45 3.4

Salida :

s 45 3.4

Entrada desde archivo:

Salida :

h 23 45.7

<< El programa ha finaliza

<< Presione enter para ce

Lectura desde el archivo

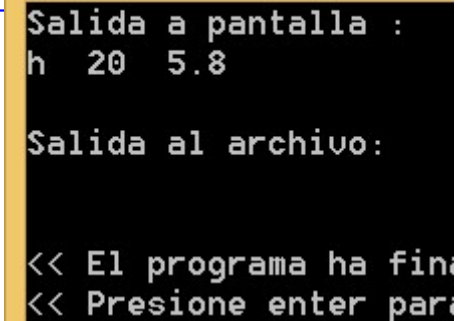
## Ejemplo 2. Un archivo de texto de salida se comporta como cout

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

int main() {
    int a; char z; double c;
    a=20; z='h'; c=5.8;
    cout << "Salida a pantalla: " << endl;
    cout << z << " " << a << " " << c << endl << endl;

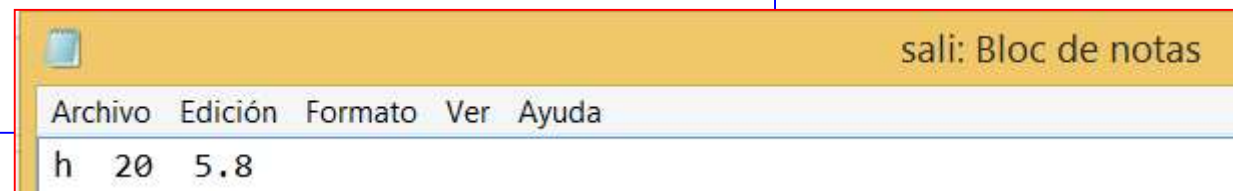
    ofstream f;
    f.open("sali.txt");

    cout << "Salida al archivo: " << endl;
    f << z << " " << a << " " << c ;
    f.close();
    return 0;
}
```



Salida a pantalla :  
h 20 5.8  
  
Salida al archivo:  
  
<< El programa ha finalizado  
<< Presione enter para continuar

Escritura en el archivo



sali: Bloc de notas

Archivo	Edición	Formato	Ver	Ayuda
h 20 5.8				

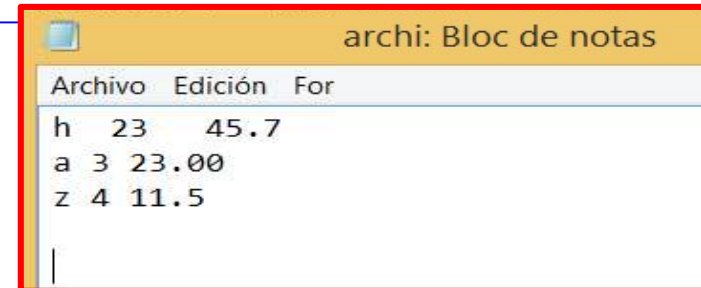
**open:** cuando es de salida si el archivo no existe lo crea, si existe lo sobrescribe vacío.

### Ejemplo 3. Se puede leer hasta fin de archivo

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

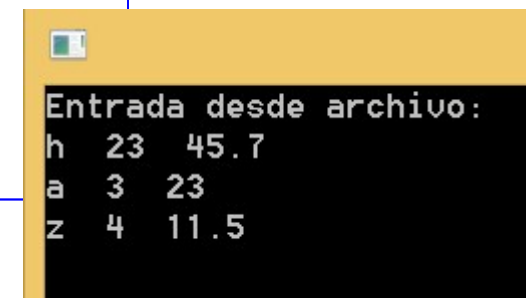
int main() {
    int a; char z; double c;
    ...
    ifstream f;
    f.open("archi.txt");

    cout << "Entrada desde archivo: " << endl;
    f >> z >> a >> c;
    while (!f.eof()){
        cout << z << " " << a << " " << c << endl;
        f >> z >> a >> c;
    }
    f.close();
    return 0;
}
```



archi: Bloc de notas

Archivo	Edición	For
h	23	45.7
a	3	23.00
z	4	11.5



Entrada desde archivo:  
h 23 45.7  
a 3 23  
z 4 11.5

## Tratamiento de Archivos

Cuando un programa quiere trabajar con un determinado archivo, debe realizar las siguientes acciones:

1. Incluir la biblioteca `<fstream>`, que contiene los elementos necesarios para procesar el fichero.
2. Usar el espacio de nombres `std`.
3. Declarar las variables que actuarán como manejadores de ficheros.
4. Abrir el flujo de datos, vinculando la variable correspondiente con el fichero especificado. Esta operación establece un vínculo entre la variable (manejador de fichero) definida en nuestro programa y el fichero gestionado por el sistema operativo.

Toda transferencia de información entre el programa y un fichero se realizará a través de la variable manejador que ha sido vinculada con dicho fichero.

5. Comprobar que la apertura del fichero se realizó correctamente. Si la vinculación con el fichero especificado no pudo realizarse por algún motivo (por ejemplo, si queremos hacer entrada de datos desde un fichero que no existe, o si es imposible crear un fichero en el que escribir datos), entonces la operación de apertura fallaría.

## Tratamiento de Archivos

6. Realizar la transferencia de información (de entrada o de salida) con el fichero a través de la variable de flujo vinculada al mismo.

Normalmente, tanto la entrada como la salida de datos se realizan mediante un **proceso iterativo**. En el **caso de entrada** dicho proceso suele requerir la **lectura de todo el contenido** del fichero.

7. Comprobar que el procesamiento del fichero del paso previo se realizó correctamente.

En el caso de procesamiento para entrada ello suele consistir en comprobar si el estado de la variable manejador indica que **se ha alcanzado el final del fichero**.

El procesamiento para salida suele consistir en comprobar si el estado de la variable manejador indica que **se ha producido un error de escritura**.

8. Finalmente, **cerrar el flujo para liberar la variable manejador** de su vinculación con el fichero.

En caso de no cerrar el flujo, éste será *cerrado automáticamente* cuando termine el ámbito de vida de la variable manejador del fichero.



## Opciones para abrir Ficheros

Para **abrir el fichero** usamos el método:

void **open** (const char\* **nombre**, int **nModo** = ios::in, int **nProt** = filebuf::openprot)

**(nombre)** es el nombre del fichero

**(nModo)** es el modo de apertura, en este caso no hace falta poner nada porque para un objeto ifstream la opción por defecto es lectura.

**(nProt)** es opcional y determina la protección del archivo.

### Modo de apertura Descripción

ios::in El fichero se abre para lectura.

ios::out El fichero se abre para escritura.

ios::app Escribe al final del fichero, lo que ya había se mantiene. (escritura secuencial)

ios::ate Se pone al final de un archivo abierto.

ios::binary El fichero es binario (si no se pone se asume que el fichero es de texto).

ios::trunc Elimina el contenido del fichero si hay algo.

ios::nocreate Al abrir, si el fichero no existe, no se crea.  
Esto genera un fallo que se puede detectar con fail  
o consultando la variable de fichero en un if.

ios::noreplace El equivalente al anterior pero con la apertura para escritura.

## Comprobar la apertura del archivo

Un fichero lógico que se haya intentado abrir y que no exista no tendrá su correspondiente objeto físico asociado.

- Para detectarlo se usa la instrucción **fail** que consulta si el fichero está en el estado correcto.

```
if (ficheroTexto.fail())  
    // el fichero no se ha abierto
```

**fail()**: Devuelve true si el archivo no se ha abierto con éxito; de lo contrario, devuelve false.

- O bien se puede hacer un **if** sobre la **variable** del tipo fichero:

```
if (!ficheroTexto)  
    // el fichero no se ha abierto
```

Ambas posibilidades son equivalentes.

## Ejemplo 4. Lectura de un archivo de texto

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

int main() {
    char caract;
    ifstream archivo_entr;
    archivo_entr.open("datos.txt");
    if (!archivo_entr)
        cout << "Error al abrir el fichero" << endl;
    else {
        archivo_entr >> caract;
        while (!archivo_entr.eof()){
            cout << caract;
            archivo_entr >> caract;
        }
    }
    archivo_entr.close();
    return 0;
}
```

datos.txt: Bloc de notas

Archivo	Edición	Formato	Ver
12 bc def ghi			
ff			
hh			
b			

```
12bcdefghiffhbb
<< El programa ha finalizado:
<< Presione enter para cerrar
```

## Lectura

Una vez que el fichero está abierto, podemos **leer elementos con el operador de entrada: “>>”**.

Se usa igual que en las instrucciones de entrada estándar (cin » variable) pero substituyendo el flujo *cin* por el *nombre de la variable del tipo ifstream*.

Para leer un carácter: **ficheroTexto » caract;**

Dicha operación pone en la variable caract el siguiente elemento leído del archivo.

**Por defecto, la lectura con el operador “>>” *ignora* los espacios en blanco, tabulaciones y saltos de línea.**

- Para que no los ignore: **ficheroTexto.unsetf(ios::skipws);**
- Para que vuelva a ignorar los espacios en blanco:  
**ficheroTexto.setf(ios::skipws);**

## Lectura: get

La instrucción ">>" se saltea los espacios en blanco.

Si se requiere **leer** todo el fichero tal y como está **sin saltarse espacios, tabulaciones, etc.**, se puede usar la función **get**.

La operación **get** se puede llamar de 2 formas:

- **get(char& car)**: Lee un sólo carácter y lo pone en la variable **car**.

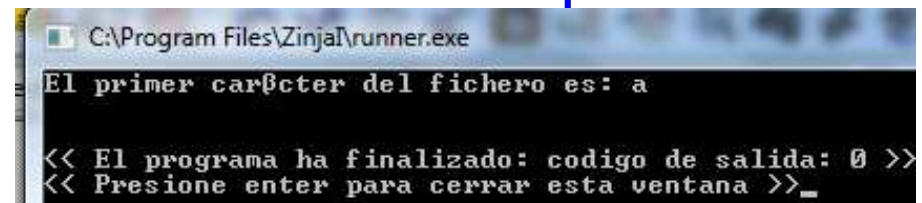
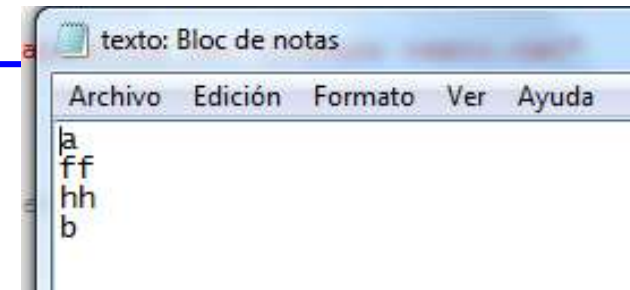
- **get (char\* cad, int numCad, char delimitador='\n' )**:

- ✓ Se usa para leer un número determinado de caracteres (indicado en *medidabuffer*) y termina si encuentra un salto de línea.
- ✓ En este caso lo que se leerá es como máximo el número de caracteres indicado en **numCad-1** o el carácter delimitador, y los almacenará en la cadena de caracteres **cad**.
- ✓ El carácter delimitador es opcional y sirve para definir un delimitador, que por defecto es el salto de línea. Si se encuentra el delimitador, **no es extraído** de la secuencia de entrada, y permanece ahí como el siguiente carácter a ser extraído.

## Ejemplo 5. Uso de get()

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

```
int main() {
    ifstream ficheroTexto;
    char c;
    ficheroTexto.open ("texto.txt");
    if (!ficheroTexto)
        cout << "Error al abrir el archivo" << endl;
    else {
        //Comprobamos que el fichero no esté vacío
        if (!ficheroTexto.eof()) {
            ficheroTexto.get(c); //Leemos el 1er caracter
            cout << "El primer caracter del fichero es: "
                 << c << endl;
        }
    }
    ficheroTexto.close();
    return 0;
}
```



## Ejemplo 6. Uso de get con parámetros

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

```
int main(){
    ifstream ficheroTexto;
    char car[30];

    ficheroTexto.open ("texto.txt");
    if (!ficheroTexto)
        cout << "Error al abrir el archivo" << endl;
    else {
        //Comprobamos que el fichero no esté vacío
        if (!ficheroTexto.eof()){
            ficheroTexto.get(car,3, '\n');
            cout << "Los caracteres leídos del fichero son: "
                << car << endl;
        }
    }
    ficheroTexto.close();
    return 0;
}
```

texto.txt: Bloc de notas

Archivo	Edición	Formato	Ver
abcd			
ff			
hh			
b			

C:\Program Files (x86)\Zinjal\bin\run

Los caracteres leídos del fichero son: ab

## Lectura: getline

**Permite la lectura de toda una línea.**

Tiene la siguiente sintaxis:

```
getline( char* cad, int numCar, char delim = '\n' );
```

Cuando se ejecuta almacena en la cadena **cad** los caracteres del fichero hasta que:

- ✓ encuentre el delimitador indicado como 3er parámetro, o
- ✓ como máximo haya leído **numCar - 1** caracteres, o
- ✓ el final del fichero

En el último elemento de la cadena **cad** se guarda **el carácter final de cadena: '\0'**.

El 3er parámetro es opcional y sirve para definir un delimitador, que por defecto es el salto de línea. Si se encuentra el delimitador, **se extrae** de la secuencia de entrada pero es descartado, **no se copia** a la cadena.



## Ejemplo 7. Uso de getline

```
int main() {  
    ifstream ficheroTexto;  
    char car[30];  
  
    ficheroTexto.open ("texto2.txt");  
    if (!ficheroTexto)  
        cout << "Error al abrir el archivo" << endl;  
  
    else {  
        if (!ficheroTexto.eof()) {  
            ficheroTexto.getline(car, '\n');  
            cout << car << endl;  
        }  
    }  
    ficheroTexto.close();  
    return 0;  
}
```

texto2.txt: Bloc de notas

Archivo	Edición	Formato	Ver
affjrpot			
hh			
b			

Los caracteres leídos del fichero son: affjrpot

<< El programa ha finalizado: código de salida: 0 >>  
<< Presione enter para cerrar esta ventana >>

## Diferencias entre get y getline

Entonces, ¿cual es la diferencia? Sutil pero importante:

- La función `get()` se detiene cuando ve el delimitador en el stream de entrada, pero no lo extrae del stream de entrada. Entonces, si se hace otro `get()` usando el mismo delimitador, retornará inmediatamente sin ninguna entrada contenida.
- La función `getline()`, por el contrario, sí extrae el delimitador del stream de entrada, pero no lo almacena en la cadena resultante.

## Diferencias entre get y getline

```
ficheroTexto.open ("texto2.txt");
if (!ficheroTexto)
    cout << "Error al abrir el archivo" << endl;
else {
    if (!ficheroTexto.eof()){
        ficheroTexto.get(car, '\n');
        cout << "Los caracteres leidos del fichero son: "
            << car << endl;
        ficheroTexto.get(car, '\n');
        cout << "Los caracteres leidos del fichero son: "
            << car << endl;
    }
}
ficheroTexto.close();
```

texto2.txt: Bloc de notas

Archivo	Edición	Formato	Ver
affjrpot hh b			

```
Los caracteres leidos del fichero son: affjrpot
Los caracteres leidos del fichero son:
<< El programa ha finalizado: codigo de salida: 0 >>
<< Presione enter para cerrar esta ventana >>
```

```
ficheroTexto.open ("texto2.txt");
if (!ficheroTexto)
    cout << "Error al abrir el archivo" << endl;
else {
    if (!ficheroTexto.eof()){
        ficheroTexto.getline(car, '\n');
        cout << "Los caracteres leidos del fichero son: "
            << car << endl;
        ficheroTexto.getline(car, '\n');
        cout << "Los caracteres leidos del fichero son: "
            << car << endl;
    }
}
ficheroTexto.close();
```

```
Los caracteres leidos del fichero son: affjrpot
Los caracteres leidos del fichero son: hh
```

```
<< El programa ha finalizado: codigo de salida: 0 >>
<< Presione enter para cerrar esta ventana >>
```

## Archivos Binarios: Las funciones read y write

```
istream& read (char* s, streamsize n);
```

### Lee un bloque de datos.

Extrae  $n$  caracteres desde el archivo de entrada y los almacena en la cadena de caracteres  $s$ .

- ✓ Para leer un tipo de dato que no sea una cadena de caracteres se deberá hacer un *casting*.
- ✓ Esta función copia un bloque de datos sin chequear su contenido (si tiene caracteres nulos o de final).

Cuando no sepamos la medida en bytes de la información que queremos leer, deberemos usar la función **sizeof** que dado un tipo nos devolverá su tamaño en bytes.

Ejemplo: tenemos una variable  $var$  de tipo *double* y queremos leer de un fichero un valor de tipo double y ponerlo en  $var$ :

```
fichero.read ( (char*)& var, sizeof(double) );
```

## Ficheros Binarios: Las funciones read y write

```
ostream& write (const char* s, streamsize n);
```

**Escribe un bloque de datos.**

Inserta en el archivo de salida  $n$  caracteres de la cadena de caracteres  $s$ .

✓ Esta función tampoco valida contenido.

## Ejemplo 8. Escritura en archivos binarios con registros

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

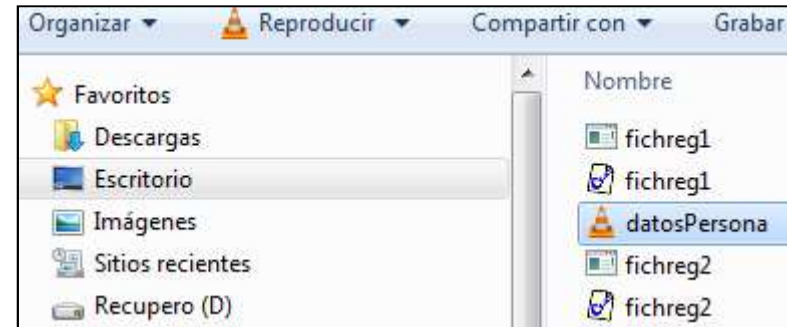
typedef char id[20];

struct persona {
    id nombre, apellido;
    int edad;
};

int main() {
    ofstream fPers;
    fPers.open("datosPersona.bin");
    if(!fPers)
        cout << "Error al abrir el fichero" << endl;
    else {
        persona p;
        strcpy(p.nombre, "Agustina");
        strcpy(p.apellido, "Soler");
        p.edad = 34;

        // Escribimos una persona
        fPers.write((char *)(&p), sizeof(p));

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



Graba en el archivo de salida una cantidad de bytes (indicadas por sizeof), obtenidos de la dirección de memoria (&p) donde están guardados los datos que se escribirán en el archivo, tratándola como una cadena de caracteres (puntero a char)

## Ejemplo 9. Lectura de archivos binarios con registros

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

typedef char id[20];
struct persona {
    id nombre, apellido;
    int edad;
};

int main() {
    persona p;
    ifstream fEntrada;
    fEntrada.open("datosPersona.bin");
    if (!fEntrada)
        cout << "Error al abrir el fichero" << endl;
    else {
        // Comprobamos que no este vacio
        if (!fEntrada.eof()) {
            fEntrada.read((char*)(&p), sizeof(p));
            cout << p.nombre << endl;
            cout << p.apellido << endl;
            cout << p.edad << endl;
        }
    }
    return 0;
}
```

```
Agustina
Soler
34

<< El programa ha finalizado:
```

Lee desde el archivo de entrada una cantidad de bytes (del sizeof), tratándola como caracteres (puntero a char) y lo almacena en la dirección de memoria del registro (p)



## Ficheros de acceso directo

- En el **acceso aleatorio (directo)**, cualquier carácter en el archivo abierto puede leerse en forma directa sin tener que leer primero en forma secuencial todos los caracteres almacenados antes que el.
- Para proporcionar acceso aleatorio a los archivos, cada objeto **ifstream** crea en forma automática un marcador de posición de archivo, **seekg/seekp**.
- Este marcador es un **numero entero largo** que representa un desplazamiento desde el principio de cada archivo e indica el **lugar desde donde se va a leer o a escribir el siguiente carácter**.



## Ficheros de acceso directo: seek

Para leer y escribir ficheros de acceso directo usaremos las mismas operaciones que ya conocemos, la diferencia es que tenemos disponible la instrucción que nos **permite posicionarnos en el fichero**:

**seekg** : “seek get” .

Se puede aplicar sobre ficheros de entrada, así como ficheros de E/S, e indica la posición del próximo get.

Para ficheros de salida, la posición para el próximo put se puede indicar con:

**seekp** : “seek put”,

ambas con el mismo formato.

Posición donde se hará la próxima lectura

**seekg( pos\_type\* posi )** → (es siempre relativa al inicio del fichero)

o bien

Cantidad *off* de posiciones a desplazarse

**seekg( off\_type\* off, ios\_base: :direcc )**

Valor	Descripción
ios::beg	El desplazamiento es relativo al inicio del fichero.
ios::end	El desplazamiento es relativo al final del fichero.
ios::cur	El desplazamiento es relativo a la posición actual en el fichero.

## Ficheros de acceso directo: seek

La función **seek** nos permite acceder a cualquier posición del fichero, no tiene por qué ser exactamente al principio de un registro.

Se hace referencia a la posición de un carácter como su **desplazamiento desde el inicio** del archivo. Por tanto, el primer carácter tiene un desplazamiento de 0, el segundo carácter tiene un desplazamiento de 1, etc., para cada carácter en el archivo.

La resolución de la función **seek** es de 1 byte.

Un desplazamiento positivo significa avanzar en el archivo y un desplazamiento negativo significa retroceder.

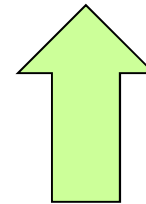
## Ficheros de acceso directo: seek

Por ejemplo:

- Si nos queremos desplazar al inicio del fichero haremos `seekg(0)`,
- Si nos queremos desplazar al final del fichero: `seekg (0, ios:: end)`.

Hay que tener en cuenta que el final de un fichero no es la posición del último elemento sino la **posición del cursor después de leer el último elemento**.

Esto quiere decir que si queremos consultar el último elemento tendríamos que ponernos en la posición: `seekg(-1, ios::end)`.



Último elemento

## Ficheros de acceso directo: tell

Para saber, en un momento determinado, **la posición en la que estamos**, ésta se puede consultar con la instrucción **tell**

**tellg ()** : devuelve la posición del cursor dentro del fichero de lectura.

Para los **ficheros de escritura** tendremos **tellp ()** que funciona igual.

## Ejemplo 10. Búsqueda en Fich. Estruct. en registros con acceso directo

```
typedef char id[20];  
struct persona {  
    id nombre, apellido;  
    int edad;  
};
```

```
int main() {  
    ifstream fEntrada;  
    int pos;  
    persona p;  
    fEntrada.open("datosPersona.bin");  
    if (!fEntrada)  
        cout << "Error al abrir el fichero" << endl;  
    else {  
        cout << "Ingrese el nro de persona a mostrar: ";  
        cin >> pos;  
        // Comprobamos que no este vacio  
        if ((pos) * sizeof(persona) > fEntrada.seekg(0, ios::end).tellg())  
            cout << "Ese elemento no existe" << endl;  
        else{  
            fEntrada.seekg((pos-1) * sizeof(persona));  
            fEntrada.read((char*)(&p), sizeof(p));  
            cout << p.nombre << endl;  
            cout << p.apellido << endl;  
            cout << p.edad << endl;  
        }  
    }  
    return 0;  
}
```

```
Ingrese el nro de persona a mostrar: 1  
Agustina  
Soler  
34
```

```
<< El programa ha finalizado: codigo de salida
```

```
Ingrese el nro de persona a mostrar: 7  
Ese elemento no existe
```

```
<< El programa ha finalizado: codigo de salida
```

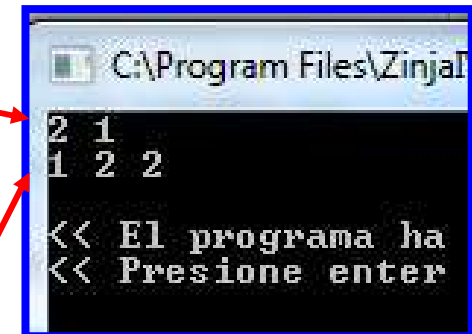
## Ejemplo 10. Búsqueda en Fich. Estruct. en registros con acceso directo

```
cout << "Ingrese el nro de persona a mostrar: ";
cin >> pos;
// Comprobamos que no este vacio
if ((pos) * sizeof(persona) > fEntrada.seekg(0, ios::end).tellg())
    cout << "Ese elemento no existe" << endl;
else{
    fEntrada.seekg((pos-1) * sizeof(persona));
    fEntrada.read((char*)(&p), sizeof(p));
    cout << p.nombre << endl;
    cout << p.apellido << endl;
    cout << p.edad << endl;
}
```

Ubicamos el puntero en la posición de la persona numero *pos*

## Ejemplo 11. Lectura y escritura de arreglos

```
int main(void) {  
    int a[3][4]={1,2,3,4},{1,1,1,1},{2,2,2,2}};  
    int b[3][4];  
    int c[12];  
    //-----  
    ofstream f;  
    f.open ("buf1.txt");  
  
    f.write ( (char*)a, sizeof(a) );  
    f.close( );  
    //-----  
    ifstream g;  
    g.open ("buf1.txt");  
  
    g.read ( (char*) b, sizeof(b) );  
    cout << b[2][3]<< " " << b[0][0] << endl;  
    g.close( );  
    // -----  
    ifstream h;  
    h.open ("buf1.txt");  
  
    h.read ( (char*) c, sizeof(c) );  
    cout << c[5] << " " << c[10] << " " << c[11];  
    h.close( );  
    return 0;  
}
```



C:\Program Files\Zinjal  
2 1  
1 2 2  
<< El programa ha  
<< Presione enter

## Ejemplo 12. Escritura de 200 pares de valores

```
int main() {  
    ofstream archi;  
    archi.open("grupo2.dat", ios::binary);  
    struct par {  
        float c1;  
        int c2;  
    };  
    par vector[200];  
  
    for (int c=1; c<=200; c++){  
        vector[c-1].c1 = rand()/(1000.0);  
        vector[c-1].c2 = rand()%1000;  
        cout << vector[c-1].c1 << " - " << vector[c-1].c2 << endl;  
    }  
    archi.write((char *)vector, sizeof(par)*200);  
    archi.close();  
    system("pause");  
    return 0;  
}
```

```
18.787 - 905  
17.958 - 391  
10.202 - 625  
26.477 - 414  
9.314 - 824  
29.334 - 874  
24.372 - 159  
11.833 - 70  
7.487 - 297  
7.518 - 177
```

```
<< El programa ha finalizado:
```



**Otros ejemplos**

## Algoritmos sobre Ficheros - Fusión de 2 fich de caracteres

```
#include <iostream>
#include <fstream>
using namespace std;
void fusionarFicheros(ifstream& f1, ifstream& f2, ofstream& fResu);

int main()
{
    char nombre1[100] , nombre2[100];
    cout << "Introduce el nombre del primer fichero: ";
    cin >> nombre1;
    ifstream f1;
    f1.open(nombre1);
    cout << "Introduce el nombre del segundo fichero: ";
    cin >> nombre2;
    ifstream f2;
    f2.open(nombre2);
    char nombreDest[100];
    cout << "Introduce el nombre del fichero destino: ";
    cin >> nombreDest;
    ofstream fDest;
    fDest.open(nombreDest);
    fusionarFicheros(f1, f2, fDest);
    f1.close ();
    f2.close ();
    fDest.close ( );
    return 0;
}
```

## Algoritmos sobre Ficheros- Fusión de 2 fich de car

```
#include <iostream>
#include <fstream>
using namespace std;
void fusionarFicheros(ifstream& f1, ifstream& f2, ofstream& fResu);

int main()
{
    char nombre1[100];
    cout << "Introduzca nombre1: ";
    cin >> nombre1;
    ifstream f1;
    f1.open(nombre1);
    cout << "Introduzca nombre2: ";
    cin >> nombre2;
    ifstream f2;
    f2.open(nombre2);
    char nombreDest[100];
    cout << "Introduzca nombreDest: ";
    cin >> nombreDest;
    ofstream fDest;
    fDest.open(nombreDest);
    fusionarFicheros(f1, f2, fDest);
    f1.close ();
    f2.close ();
    fDest.close ();
    return 0;
}

void fusionarFicheros(ifstream& f1, ifstream& f2, ofstream& fDest)
{
    char e;
    f1.unsetf(ios::skipws);
    // Ponemos enfDest el contenido del primer fichero
    while (f1 >> e)
        fDest << e;
    f2.unsetf(ios::skipws);
    // Ponemos enfDest el contenido del segundo fichero
    while (f2 >> e)
        fDest << e;
}
```

## Algoritmos sobre Ficheros- Fusión de 2 fich de car

```
void fusionarFicheros(ifstream& f1, ifstream& f2, ofstream& fDest)
{
    char e;
    f1.unsetf(ios::skipws);
    // Ponemos enfDest el contenido del primer fichero
    while (f1 >> e)
        fDest << e;
    f2.unsetf(ios::skipws);
    // Ponemos enfDest el contenido del segundo fichero
    while (f2 >> e)
        fDest << e;
}
```

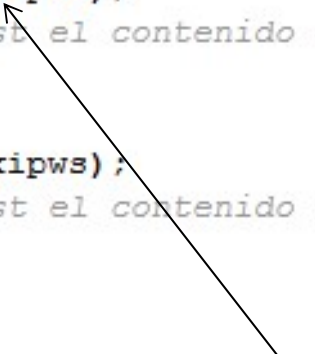
Cualquier bandera establecida puede desactivarse.  
Para desactivar una bandera usamos la función **unsetf**.

Por ejemplo, lo que sigue hará que el programa deje de incluir signos de más antes de los enteros positivos que se envían al flujo cout:

**cout.unsetf(ios::showpos);**

## Algoritmos sobre Ficheros- Fusión de 2 fich de car

```
void fusionarFicheros(ifstream& f1, ifstream& f2, ofstream& fDest)
{
    char e;
    f1.unsetf(ios::skipws);
    // Ponemos enfDest el contenido del primer fichero
    while (f1 >> e)
        fDest << e;
    f2.unsetf(ios::skipws);
    // Ponemos enfDest el contenido del segundo fichero
    while (f2 >> e)
        fDest << e;
}
```



Cuando la bandera de formato **skipws** se establece, los espacios en blanco seguidos se leen y se descartan del flujo hasta leer algo que no sea blanco.

Esto se aplica a todas las operaciones de entrada con formato realizado con el operador >> en la secuencia.

Espacios de tabulación, de retornos de carro y espacios en blanco son todos considerados espacios en blanco (ver isspace).

Para flujos estándares, la bandera skipws se encuentra en la inicialización.

## Ejercicios con pares de números

1. Mediante un programa C++, escribir un archivo binario llamado, **grupo.dat**, formado por un conjunto de 200 pares de números generados aleatoriamente. Cada par de datos se conforma por un flotante y un entero.
2. Escribir un programa que abra el archivo generado en el ejercicio anterior, y solicite al usuario que ingrese un flotante, un entero y una posición.

El programa debe **sobreescribir** el par **en la posición** ingresada por el usuario, por el nuevo par.

Luego, debe mostrar la lista de datos por pantalla mostrando un par por línea.

1.

```
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <cstdlib>
using namespace std;

int main() {
    ofstream archi;
    archi.open("grupo.dat", ios::binary);
    float f; int i;

    for (int c=1; c<=200; c++){
        f=rand()/(1000.0);
        i=rand()%1000;
        cout << f << " - " << i << endl;
        archi.write((char *)&f, sizeof(f));
        archi.write((char *)&i, sizeof(i));
    }
    archi.close();

    system("pause");
}
```



2.

```
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <cstdlib>
using namespace std;

int main(){
    fstream archi;
    archi.open("grupo.dat", ios::binary | ios::in | ios::out);
    float f,nf; int i,ni, posicion;

    cout << "Posicion a sobrescribir (en 0..199) ? ";
    cin >> posicion;
    cout << "Nuevo float ? ";
    cin >> nf;
    cout << "Nuevo int ";
    cin >> ni;

    archi.seekg(posicion*(sizeof(f)+sizeof(i)),ios::beg);

    archi.read((char *) &f, sizeof(float));
    archi.read((char *) &i, sizeof(int));

    cout<<"Par de valores de la posicion " << posicion << " : " <<f<<" "<<i<<endl;

    archi.seekp(posicion*(sizeof(f)+sizeof(i)),ios::beg);
    archi.write((char *)&nf, sizeof(float));
    archi.write((char *)&ni, sizeof(int));

    archi.close();

    system("pause");
}
```



## Ejercicio de transacciones



### Funcionalidad:

- actualiza las cuentas existentes,
- agrega nuevas cuentas,
- elimina cuentas, y
- guarda un listado con formato de todas las cuentas actuales en un archivo de texto

```
#include <stdio.h>
#include <fstream>
#include <iostream>
#include <iomanip>
using namespace std;

// clientData structure definition
- struct clientData {
    unsigned int acctNum; // account number
    char lastName[ 15 ]; // account last name
    char firstName[ 10 ]; // account first name
    double balance; // account balance
};
typedef struct clientData cliente;

// prototypes
unsigned int enterChoice();
void textFile (fstream &);
void updateRecord( fstream & );
void newRecord( fstream & );
void deleteRecord( fstream & );
void mostrarLinea (ostream&, const cliente &);
int obtenerCuenta (const char * );
```

```
int main( void )
{
    fstream credito;
    credito.open("credit.dat", ios::in|ios::out);
    if ( !credito ) {
        cout << "File could not be opened." ;
    }
    else {
        unsigned int choice; // user's choice// enable user to specify action
        while ( ( choice = enterChoice() ) != 5 ) {
            switch ( choice ) {
                case 1:
                    textFile( credito );break;
                case 2:
                    updateRecord( credito );break;
                case 3:
                    newRecord( credito );break;
                case 4:
                    deleteRecord( credito );break;
                default:
                    cout << "Incorrect choice" ;break;
            }
        }
        credito.close();
    }
} // end main
```

```
unsigned int enterChoice( void )  
{  
    unsigned int menuChoice;  
    cout <<"\nEnter your choice\n"  
        <<"1 - store a formatted text file of accounts called\n"  
        <<"    \"accounts.txt\" for printing\n"  
        <<"2 - update an account\n"  
        <<"3 - add a new account\n"  
        <<"4 - delete an account\n"  
        <<"5 - end program\n? " ;  
  
    cin >> menuChoice;  
    return menuChoice;  
}
```

```
// create formatted text file for printing
```

```
void textFile( fstream & leerDeArchivo)
```

```
{
```

```
    ofstream salida; // accounts.txt file pointer
```

```
    cliente client ;
```

```
    salida.open( "accounts.txt", ios::out);
```

```
    if (!salida){
```

```
        cout << "File could not be opened." ;
```

```
    }
```

```
    else {
```

```
        salida << left << setw(10) << "Acct"<<setw(16) <<"Last Name"  
        << setw(14)<<"First Name"<< right <<setw(10) <<"Balance"<<endl;
```

```
        leerDeArchivo.seekg (0);
```

```
        leerDeArchivo.read((char *) &client, sizeof(cliente));
```

```
        while ( !leerDeArchivo.eof() ) {
```

```
            mostrarLinea (salida,client);
```

```
            leerDeArchivo.read((char *) &client, sizeof(cliente));
```

```
        }
```

```
        leerDeArchivo.close();
```

```
    }
```

```
} // end function textFile
```

Se escriben  
los títulos  
del listado



```

void mostrarLinea( ostream & salida, const cliente& registro)
{
    salida << left << setw(10) << registro.acctNum<<setw(16) <<registro.lastName
        << setw(11)<< registro.firstName << right <<setw(10) <<registro.balance<<endl;
}

int obtenerCuenta (const char * indicador)
{
    int num;
    do {
        cout << indicador <<"(1-100): ";
        cin >> num;
    } while (num <1 || num > 100);
    return num;
}

```

```
// update balance in record
```

```
void updateRecord( fstream & actualizarArch )
```

```
{
```

```
    unsigned int account = obtenerCuenta ("Escriba la cuenta que desea actualizar");
```

```
    double transaction;
```

```
    // create clientData with no information
```

```
    cliente client = { 0, "", "", 0.0 };
```

```
    actualizarArch.seekg ( ( account - 1 ) * sizeof(cliente));
```

```
    actualizarArch.read((char *) &client, sizeof(cliente));
```

```
    if ( client.acctNum != 0 ) {
```

```
        mostrarLinea (cout,client);
```

```
        cout <<endl <<"Enter charge ( + ) or payment ( - ): " ;
```

```
        cin >>transaction ;
```

```
        client.balance += transaction; // update record balance
```

```
        actualizarArch.seekp ( ( account - 1 ) * sizeof(cliente));
```

```
        actualizarArch.write((char *) &client, sizeof(cliente));
```

```
        mostrarLinea (cout,client);
```

```
    }
```

```
    else cout << "La cuenta " << account <<" no tiene información" << endl;
```

```
    // create and insert record
```

```
}
```



```

void newRecord( fstream & insertarEnArch)
{
    unsigned int account = obtenerCuenta ("Escriba la cuenta que desea actualizar");
    insertarEnArch.seekg ( ( account - 1 ) * sizeof(cliente));
    cliente client ;
    insertarEnArch.read((char *) &client, sizeof(cliente));
    if ( client.acctNum != 0 ) {

        cout<< " Escriba apellido, nombre y saldo" << endl;
        cin >> client.lastName >> client.firstName>> client.balance ;

        insertarEnArch.seekp ( ( account - 1 ) * sizeof(cliente));
        insertarEnArch.write((char *) &client, sizeof(cliente));
    }
    else cout << "La cuenta " << account <<" ya tiene informacion" << endl;
}

```



```

void deleteRecord( fstream & eliminarEnArch)
{
    unsigned int account = obtenerCuenta ("Escriba la cuenta que desea eliminar");
    eliminarEnArch.seekg ( ( account - 1 ) * sizeof(cliente));
    cliente client ;
    eliminarEnArch.read((char *) &client, sizeof(cliente));
    if ( client.acctNum != 0 ) {
        cliente client = { 0, "", "", 0.0 };
        eliminarEnArch.seekp ( ( account - 1 ) * sizeof(cliente));
        eliminarEnArch.write((char *) &client, sizeof(cliente));
        cout << " La cuenta"<<account << "ha sido eliminada";
    }
    else cout << "La cuenta " << account <<" esta vacia" << endl;
}

```

**LEER**

## **Capítulo 8**

**“Flujos de archivos de E/S y archivos de datos”**

**Libro: “C++ para ingeniería y ciencias” 2da edición -**

**Gary J. Bronson, pág. 443**