Francisco Grimaldo Moreno
Ariadna Fuertes Seder

Tema 5
Archivos o Ficheros

Informática
Grado en Física
Universitat de València

Ariadna.Fuertes@uv.es
Francisco.Grimaldo@uv.es

Guión del tema

- Introducción:
  - Concepto de Fichero
  - Tipos de acceso a ficheros
  - Tipos de ficheros: tipo texto y tipo binario
  - Ficheros lógicos y ficheros físicos
- Operaciones sobre ficheros
  - Apertura del fichero
  - Cierre del fichero
  - Escritura en fichero
  - Lectura de fichero
  - Otras instrucciones
- Procesamiento de un fichero
- Operaciones de lectura y de escritura en ficheros de texto
  - Escritura mediante <<
  - Lectura mediante >>
  - Lectura mediante `get()`
  - Lectura de estructuras
  - Paso como parámetro de un fichero a una función
- Operaciones de lectura y de escritura en ficheros binarios
  - Método de acceso directo
  - Lectura y escritura de ficheros binarios

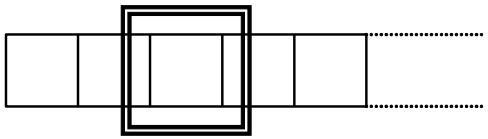
Introducción (1/6)

- **Concepto de fichero**
- Todas las estructuras de datos que hemos visto hasta ahora utilizan memoria principal.
- Esto tiene dos limitaciones importantes:
  - 1. Los datos desaparecen cuando el programa termina.
  - 2. La cantidad de los datos no puede ser muy grande debido a la limitación de la memoria principal.
- Por eso existen también estructuras especiales que utilizan memoria secundaria: los ficheros.
- El fichero es además una estructura dinámica, en el sentido de que su tamaño puede variar durante la ejecución del programa dependiendo de la cantidad de datos que tenga.

Introducción (2/6)

- **Tipos de acceso a ficheros**

- Al estar en memoria secundaria, no todos los elementos del fichero son accesibles de forma inmediata. Solamente se puede acceder cada vez a un único elemento del fichero, que se denomina *ventana del fichero*.



- Dependiendo de cómo se desplaza la ventana por el fichero, podemos distinguir dos tipos de acceso:
  - Acceso secuencial: La ventana del fichero sólo puede moverse hacia delante a partir del primer elemento y siempre de uno en uno.
  - Acceso directo: La ventana del fichero se puede situar directamente en cualquier posición del fichero. Es un acceso similar al utilizado en los arrays.

Introducción (3/6)

- El acceso directo suele ser más eficiente, ya que para leer un dato no hace falta leer antes todos los anteriores.

- La razón por la que existe el acceso secuencial es que existen dispositivos de memoria secundaria que sólo admiten acceso secuencial (como por ejemplo las cintas). Además, el acceso secuencial se utiliza también en dispositivos que admiten acceso directo cuando queremos leer los elementos de forma secuencial, ya que este acceso es más sencillo.

- Para trabajar con ficheros (memoria secundaria) serán necesarias unas operaciones básicas:

  - para poder escribir la información en el fichero
  - y para poder recuperarla y ponerla en memoria,

  así como unas funciones para abrir el fichero y para cerrarlo.

Introducción (4/6)

- **Tipos de ficheros**
- Existen dos tipos principales de ficheros (en función de la manera en que se guarda la información dentro de él):
- Ficheros de tipo texto
- Se guarda la información en forma de secuencia de caracteres (tal y como se mostraría por pantalla). Por ejemplo, el valor entero ‘25’ se transformará en los caracteres ‘2’ y ‘5’.
- Por este motivo las operaciones de escritura y lectura de ficheros de este tipo, serán similares a las utilizadas para escribir en pantalla y leer de teclado.
- Ficheros de tipo binario
- Contiene secuencias de elementos de un tipo determinado de datos. Es decir, la información se guarda en el fichero tal y como está en memoria principal: compuesta por unos y ceros. Por ejemplo, el valor entero ‘25’ se guardará como ‘00011001’.
- Un fichero binario es por tanto similar a un vector. Para acceder a estos datos necesitaremos unas funciones específicas que serán diferentes a las de los ficheros de texto.

Introducción (5/6)

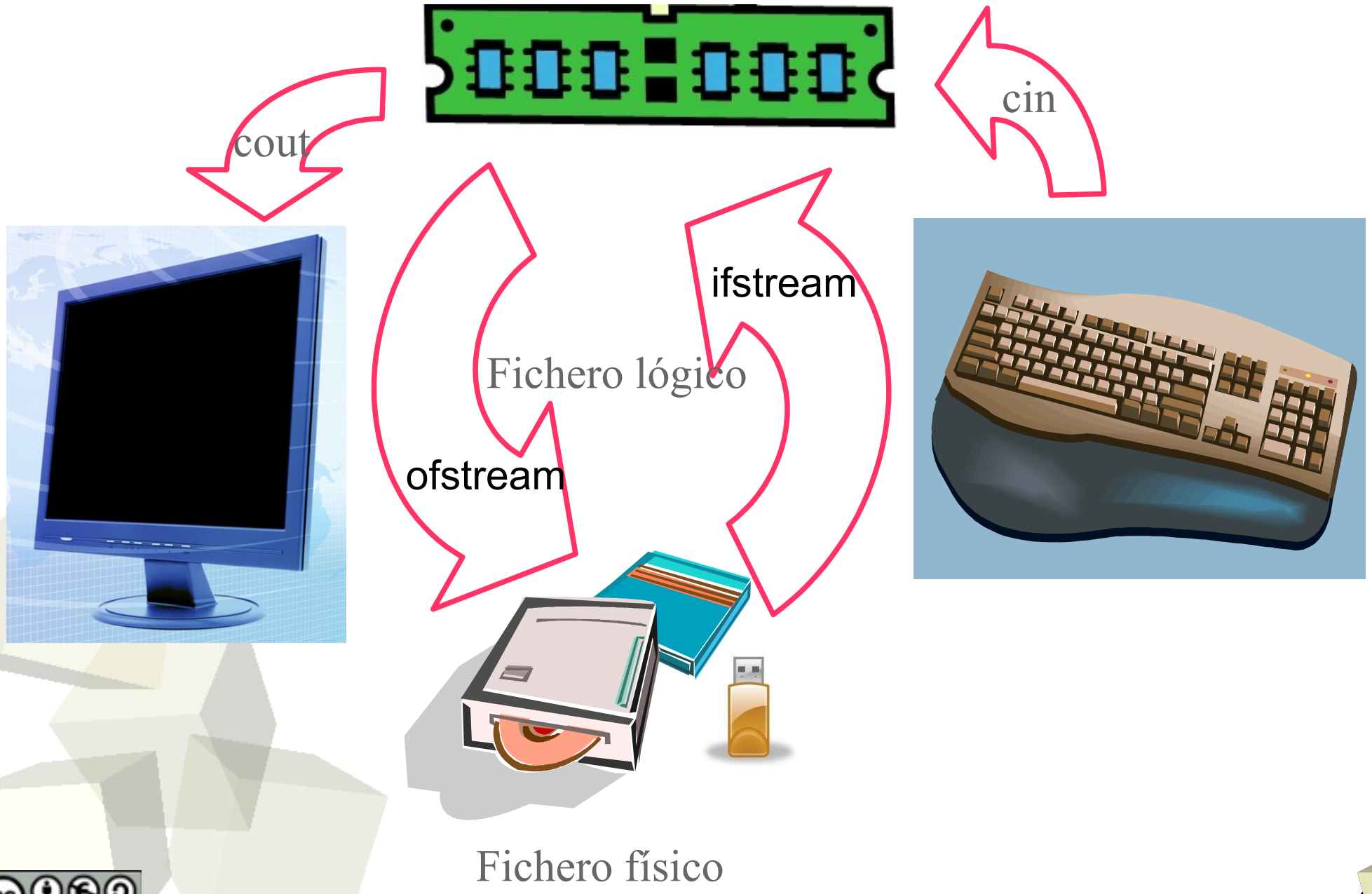
- **Ficheros lógicos y ficheros físicos**
- En un lenguaje de programación, los ficheros son un tipo de dato más, y un fichero concreto se referencia utilizando una variable de tipo fichero. Es lo que denominamos fichero lógico.
- En C++ existen dos tipos de datos básicos para declarar ficheros:
  - ifstream // Para declarar ficheros de entrada (in) (de donde se lee)
  - ofstream // Para declarar ficheros de salida (out) (donde se escribe)
- Para utilizar estos tipos hay que incluir antes el fichero de cabecera <fstream>

- **Ejemplo** de sentencia que declara una variable (fichero lógico) de tipo fichero de salida.
  ofstream f;

- Pero esta variable, para que nos sea de utilidad tiene que estar asociada con un fichero "real", es decir, por un fichero reconocido por el sistema operativo (por ej. "datos.txt") puesto que al final será el sistema operativo quien realice la escritura o lectura de ese fichero. Este fichero es lo que se denomina fichero físico.
- Para relacionar el fichero lógico con el fichero físico necesitamos realizar una operación de *apertura del fichero*.

Introducción (6/6)

- **Resumen**



Operaciones sobre ficheros (1/5)

- **Apertura del fichero:**
- *Físicamente*, un fichero es un conjunto de bits guardados en un determinado soporte (generalmente magnético) y al que se accede mediante una referencia al soporte (unidad) y a la localización sobre el soporte (por ejemplo: Sector, cilindro, cara,... en el caso de soporte magnético sobre disco.)
- La *definición lógica* del fichero es un conjunto de información útil para el usuario, y guardada con un nombre (al que se puede acceder a través de un cierto camino o 'path').
- La manera de relacionar ambos conceptos es a través del sistema operativo, y en los programas realizados por los usuarios a través de la operación de apertura de ficheros.
- La operación de apertura de fichero utiliza la información que conoce el usuario para relacionarla con la descripción física y preparar un descriptor que utilizará el programa para acceder a la información contenida en el fichero.
- En C++ esta operación se realiza con la instrucción open:
  nombre_fichero_logico.open (nombre_fichero_fisico);

Operaciones sobre ficheros (2/5)

- Ejemplo:
  f.open("datos.txt");

- A partir de ese momento ya podemos utilizar el fichero.
- Hay que tener en cuenta que, por defecto, los ficheros de entrada (ifstream) se abren sólo para lectura poniendo la ventana del fichero en el primer elemento del fichero. Además el fichero debe existir, si no se genera un error.
- Por defecto, los ficheros de salida (ofstream) se abren sólo para escritura creando el fichero nuevo. Si el fichero ya existía es borrado.
- Para modificar el modo de apertura se puede añadir un parámetro más a la instrucción open, que indica el modo de apertura. Sólo vamos a ver el modo ios::app que sirve para abrir un fichero de salida en modo añadir, de manera que no borra el fichero y la ventana del fichero se pone después del último elemento. Si el fichero no existe da error.
- nombre_fichero_logico.open (nombre_fichero_fisico, ios::app);

Operaciones sobre ficheros (3/5)

- Ejemplo:
  f.open("datos.txt" , ios::app);

- Para saber si la apertura del fichero ha dado error se utiliza el operador ! aplicado al fichero:

- Ejemplo:
  ```
  if (!f)
      cout << "Error abriendo el fichero" << endl;
  ```

- Esta condición se debe poner siempre que abramos un fichero, puesto que si ha habido un error, no se podrá realizar ninguna operación con él.

Operaciones sobre ficheros (4/5)

- **Cierre del fichero:**
- Cuando se han acabado de realizar todas las operaciones, SIEMPRE hay que cerrar el fichero. Esta operación destruye las estructuras que se han creado para abrirlo (tanto del programa como del sistema operativo). También actualiza totalmente el fichero, escribiendo toda la información que pudiera quedar en el buffer (ya que normalmente la información pasa a través de un buffer).

- Para cerrar el fichero se utiliza la instrucción close:
  nombre_fichero_logico.close ();

- Ejemplo:
  f.close();

Operaciones sobre ficheros (5/5)

- **Escritura en fichero:**
- Las funciones de escritura en fichero se utiliza para poner información que está en la memoria del ordenador, en un fichero, para poder utilizarla posteriormente. La manera de escribir en el fichero dependerá del tipo de fichero en el que queremos guardar la información: texto o binario.
- En los siguientes puntos desarrollaremos los dos casos.
- **Lectura de fichero:**
- Las funciones de lectura en ficheros se utilizan para recuperar información guardada en fichero y ponerla en memoria para poder trabajar directamente con ella.
- Al igual que en la escritura dependerá del tipo de fichero (si de texto o binario) las funciones son unas u otras.
- En cualquier caso, para poder leer información de un fichero, hay que saber exactamente cómo ha sido guardada en él.
- **Otras instrucciones:**
- El carácter EOF es el carácter de final de fichero. Para poder averiguar si se ha llegado a leer este carácter, tenemos la función eof () que detecta si se ha llegado al final de fichero.
- eof () nos devuelve verdadero si estamos en el final de fichero. Falso en cualquier otro caso.

Procesamiento de un fichero

- Siempre que trabajemos con ficheros, la primera tarea que tendremos que realizar será abrir el fichero y relacionarlo adecuadamente con un descriptor, que será, a partir de ese momento, la referencia que utilizaremos para trabajar con el fichero.

- En ese momento ya se podrá realizar cualquier operación con los ficheros y finalmente lo cerraremos.

- Resumiendo, para trabajar con ficheros deberemos seguir el siguiente esquema:

  Apertura de Fichero → Operaciones de lectura o escritura → Cierre del fichero

Operaciones en ficheros de texto (1/15)

- Las operaciones que se pueden realizar sobre un fichero de texto son exactamente las mismas que se pueden realizar sobre cin (para ficheros de entrada) y cout (para ficheros de salida). De hecho cin y cout son ficheros predefinidos, que están asociados con la entrada estándar y la salida estándar del sistema operativo que normalmente son el teclado y la pantalla respectivamente.

- Ejemplo: Para escribir el numero 10 en el fichero f:

```
f << 10;
```

Para escribir un string:

```
f << "Hola";
```

- Escritura mediante <<
- Sirve para poner el contenido de una variable x (de cualquier tipo simple) en un fichero:

```
f << x;
```

- Lectura mediante >>
- Para leer algo de un fichero y almacenarlo en un variable x:

```
f >> x;
```

Operaciones en ficheros de texto (2/15)

- **Ejemplo**: Programa para escribir los números del 1 al 10 en el fichero datos.txt

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

int main()
{
    ofstream f;    //fichero de salida
    int i;

    f.open("datos.txt");  // Apertura del fichero
    if ( !f )
        cout << "Error abriendo el fichero" << endl;
    else        // Operaciones sobre el fichero
    {
        for(i = 1; i <= 10; i++)
           f << i << endl;      //escribe el contenido de i y salta 1 línea
        f.close();  // Cierre del fichero
    }
    return 0;
}
```

Operaciones en ficheros de texto (3/15)

- **Ejemplo**: Programa para leer los 10 números enteros del fichero y mostrarlos por pantalla.

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

int main()
{
    ifstream f;
    int i, dato;

    f.open("datos.txt");
    if ( !f )
        cout << "Error abriendo el fichero" << endl;
    else
    {
        for(i = 1; i <= 10; i++)
        {
            f >> dato;
            cout << dato << endl;
        }
        f.close();
    }
    return 0;
}
```

Operaciones en ficheros de texto (4/15)

- Sin embargo, lo normal es que no sepamos cuantos elementos vamos a leer, sino que queremos leer hasta que lleguemos al final del fichero. Para ello se puede utilizar un bucle while de la siguiente forma:

```
while (f >> dato)
    cout << dato << endl;
```

- Cuando una instrucción para leer de fichero acaba con éxito, devuelve cierto, y cuando se produce algún tipo de error (entre los que se incluye llegar al final del fichero), devuelve falso. De esta forma, la instrucción anterior leerá, mientras sea posible, todos los números del fichero.

- Esta forma de leer del fichero se puede utilizar con cualquier tipo de lectura: con >> y también con getline si queremos leer hasta el salto de línea.

```
getline (variable_ifstream , variable_tipo_string );

while ( getline ( f , datoTipoString)
    cout << datoTipoString << endl;
```

- El programa anterior modificado de forma que incluya la instrucción while quedará de la siguiente forma:

Operaciones en ficheros de texto (5/15)

- **Ejemplo**: Programa para leer los números enteros de un fichero y mostrarlos por pantalla.

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

int main()
{
    ifstream f;
    int dato;

    f.open("datos.txt");
    if ( !f )
        cout << "Error abriendo el fichero" << endl;
    else
    {
        while(f >> dato)
            cout << dato << endl;

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

Operaciones en ficheros de texto (6/15)

- Lectura mediante get()

- Si queremos leer el fichero carácter a carácter, lo más normal será utilizar el método get( ), sin embargo éste no devuelve cierto o falso para saber si se ha podido leer con éxito, puesto que tiene que devolver el carácter que ha leído.

- La forma de leer un fichero con get() será por tanto ligeramente distinta a la que hemos visto. Deberemos usar el *método eof().*

- El método eof() es cierto si el dato que hemos leído era un final de fichero y falso en caso contrario.

- Por esta razón hay que leer primero el carácter y después comprobar si hemos llegado a final de fichero.

Operaciones en ficheros de texto (7/15)

- **Ejemplo**: Programa para leer los caracteres de un fichero y mostrarlos por pantalla.

```
#include <iostream>
#include <fstream>

int main()
{
    ifstream f;
    char dato;

    f.open("datos.txt");
    if ( !f )
        cout << "Error abriendo el fichero" << endl;
    else
    {
        dato = f.get();
        while( ! f.eof() )
        {
            cout << dato << endl;
            dato = f.get();
        }
        f.close();
    }
    return 0;
}
```

Operaciones en ficheros de texto (8/15)

- Lectura de estructuras

- Cuando queremos leer datos simples de un fichero, se puede realizar fácilmente introduciendo la lectura dentro de la condición de un bucle while.

- Sin embargo, para leer una estructura se deben leer primero cada uno de sus campos antes de considerar la lectura correcta, por lo que para poder efectuar la lectura en la condición del while, habrá que definir una función que realice dicha lectura y devuelva true si se ha podido realizar sin errores y false en caso contrario.

- No obstante, si el fichero ha sido escrito por nuestra propia aplicación y hemos verificado que la estructura es correcta antes de escribirlo, podemos asumir que: “si existe el primer campo de la estructura, también existirán los siguientes” por lo que podemos poner el primer campo dentro del bucle while y no usar el método eof().

- Veamos primero como sería la lectura sin usar funciones y en el apartado siguiente modificaremos el programa, incluyendo funciones, de manera que sea más correcta su implementación.

Operaciones en ficheros de texto (9/15)

- Ejemplo con método eof() y sin funciones:

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

struct Telefono
{
    string nombre;
    int telefono;
};

int main(void)
{
    Telefono tel;
    ifstream f;

    f.open("guia.dat");
    if( !f )
        cout << "Error abriendo el fichero" << endl;
    else
    {
                getline(f, tel.nombre); //Lee del fichero una línea de texto
                f >> tel.telefono; //Lee el entero que corresponde al núm.
                f.ignore(); //Ignora un carácter del buffer que es el '\n'

                while( !f.eof() )
                {
                        cout << tel.nombre << endl << tel.telefono << endl;

                        getline(f, tel.nombre); //Lee del fichero
                        f >> tel.telefono; //Lee el entero
                        f.ignore(); //Ignorar el '\n'
                }
                f.close();
        }
        return 0;
}
```

- Ejemplo sabiendo que la escritura en fichero ha sido correcta y sin funciones:

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

struct Telefono
{
    string nombre;
    int telefono;
};

int main(void)
{
    Telefono tel;
    ifstream f;

    f.open("guia.dat");
    if( !f )
        cout << "Error abriendo el fichero" << endl;
    else
    {
                //Lee del fichero una línea de texto (si existe)
                while( getline(f, tel.nombre) )
                {
                        //Si entra en el bucle es porque hay dato completo
                         //Lee el entero que corresponde al número
                        f >> tel.telefono;
                        f.ignore(); //Ignorar el '\n'
                        cout << tel.nombre << endl << tel.telefono << endl;
                }
                f.close();
        }
        return 0;
}
```

Operaciones en ficheros de texto (10/15)

- Paso como parámetro de un fichero a una función

- Para terminar conviene observar que los ficheros se han de pasar siempre como parámetros por referencia, no importa si los vamos a modificar o no.

- Además para este caso hemos usado la función *getline*, que nos permite leer una línea de fichero completa. Es evidente que para que esto funciones el fichero de datos debe incluir en cada línea un dato distinto (en una línea un nombre, en la siguiente el teléfono asociado y así sucesivamente).

- Si modificamos el programa anterior incluyendo una función F_IntroTel que se encargará de leer la información del fichero y devolvernos un booleano indicando si hemos llegado al final de fichero o no.

Operaciones en ficheros de texto (11/15)

- Ejemplo con método eof() y funciones:

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

struct Telefono
{
    string nombre;
    int telefono;
};

//Prototipos de las funciones a usar
void EscribeTel (Telefono  ) ;
bool F_IntroTel (ifstream & , Telefono &  ) ;

int main(void)
{
    Telefono tel;
    ifstream guia;

    guia.open("guia.dat");
    if(!guia)
        cout << "Error abriendo el fichero" << endl;
    else
    {
        while(F_IntroTel(guia, tel))
        {
            EscribeTel(tel);
            cout << endl;
        }
        guia.close();
    }
    return 0;
}
```

```
void EscribeTel (Telefono tel)
{
        cout << tel.nombre << endl << tel.telefono ;
        cout << endl << endl;
}

bool F_IntroTel(ifstream & f, Telefono & tel)
{
        //Lee del fichero una línea de texto
        getline( f , tel.nombre);

        //Lee el entero que corresponde al número
        f >> tel.telefono;

        //Ignora un carácter del buffer que es el '\n'
        f.ignore();

        return ( !f.eof() );
}
```

Operaciones en ficheros de texto (12/15)

- Ejemplo de programa que lee de fichero:
- Tenemos un fichero donde se guarda información acerca de los componentes mecánicos de automóvil. La ficha de cada componente mecánico tiene la información sobre:
  - el nombre de la pieza (pueden ser muchas palabras)
  - unidades disponibles (es un número entero)
  - precio de la pieza (puede tener decimales)

  La estructura del fichero (“autos.txt”) es la siguiente:
  - En la primera línea del fichero nos dice cuántas piezas hay en el fichero.
  - En la siguiente línea, el nombre de la pieza y luego,
  - en otra línea nos encontramos las unidades disponibles y su precio.
  - Ejemplo:
    - 3
    - correa de distribucion de ford mondeo 1.4
    - 24 123.85
    - disco de freno de audi
    - 12 12.4
    - disco de freno de opel corsa
    - 30 3.70
- Se trata de hacer un programa que lea la información del fichero y nos la almacene en memoria, mostrándonos a continuación esa información.
- En el programa debe ser capaz de almacenar un máximo de 500 piezas. (El fichero nunca tendrá más de ese número y estará escrito correctamente).

Operaciones en ficheros de texto (13/15)

- Solución:

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

#define TAM 500 //numero maximo de piezas que tendremos
struct pieza
{
        string nombre;
        int unidades;
        float precio;
};
void LeerFichero  (ifstream &f,pieza vector[TAM] , int & num );
void MostrarMemoria  (pieza vector[TAM] , int  num);

void LeerFichero  (ifstream &f, pieza vector[TAM] , int & num )
{
        int i;
        f >> num;
        f.ignore();
        for ( i=0 ; i < num ; i++)
        {
                getline( f , vector[i].nombre);
                f >> vector[i].unidades  >> vector[i].precio ;
                f.ignore();
        }
}
void MostrarMemoria  (pieza vector[TAM] , int  num)
{
        int i;
        for ( i=0 ; i < num ; i++)
        {
                cout << vector[i].nombre << endl;
                cout << vector[i].unidades <<" " << vector[i].precio ;
        }
}
```

```
int main ( void )
{
   pieza lista[TAM];
   int cuantos; //numero de piezas que tenemos
   string nombre;
   ifstream fich;

   cout << "Este programa lee el fichero de piezas.\n" ;
   cout << "Dame el nombre del fichero \n" ;
   cin >> nombre;

   fich.open( nombre.c_str());

  if( !fich )
        cout << "Error abriendo el fichero\n";
   else
   {
      LeerFichero ( fich, lista , cuantos );
      fich.close();

      MostrarMemoria (lista , cuantos );
   }

   system("pause");
   return 0;
}
```

Operaciones en ficheros de texto (14/15)

- Ejemplo de programa que lee de fichero y escribe en fichero:

- Vamos a hacer una nueva versión del programa anterior y le vamos a añadir otra función que nos permita recorrer toda la información que hay en memoria y la guarde en el fichero (es decir, sobrescriba el fichero anterior).

- Consideraciones:

- Hay que tener en cuenta que debemos escribir el fichero según el formato marcado anteriormente o luego no seremos capaces de leerlo correctamente. (si hemos hecho así la función MostrarMemoria, la de escribir en fichero se reduce a cambiar el cin por el nombre de la variable de acceso a fichero).

- Cuando vayamos a hacer uso de lectura y escritura en el mismo fichero, y para evitar problemas en los tiempos de acceso, se recomienda abrir el fichero dentro de la propia función de lectura/escritura, siendo estas la que devuelvan si ha sido capaz de abrirlo o no.

- Vamos a rehacer el programa anterior teniendo en cuenta esta consideración.

Operaciones en ficheros de texto (15/15)

- Solución:

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
#define TAM 500 //numero maximo de piezas que tendremos
struct pieza
{
        string nombre;
        int unidades;
        float precio;
};
bool LeerFichero  ( string , pieza [TAM] , int &  );
bool EscribirFichero  ( string , pieza [TAM] , int  );
bool LeerFichero  (  string nom, pieza vector[TAM] , int & num )
{
        int i;
        bool error = false;
        ifstream f;

        f.open( nom.c_str());
        if( ! f )
                error = true ;
        else
        {
                f >> num;
                f.ignore();
                for ( i=0 ; i < num ; i++)
                {
                        getline( f , vector[i].nombre);
                        f >> vector[i].unidades  >> vector[i].precio ;
                        f.ignore();
                }
                fich.close();
        }
        return error;
}
bool EscribirFichero(string nom, pieza vector[TAM], int num)
{
        int i;
        bool error = false;
        ofstream g;

        g.open( nom.c_str());
        if( ! g )
                error = true ;
        else
        {
                g << num << endl;
                for ( i=0 ; i < num ; i++)
                {
                        g << vector[i].nombre << endl ;
                        g << vector[i].unidades <<" " ;
                        g << vector[i].precio << endl ;
                }
                g.close();
        }
        return error;
}
int main ( void )
{
  pieza lista[TAM];
  int cuantos;  //numero de piezas que tenemos
  string nombre;

  cout << "Dame el nombre del fichero \n" ;
  cin >> nombre;

  if( LeerFichero ( nombre, lista , cuantos ) == false )
  {
     if( EscribirFichero ( nombre, lista , cuantos ) )
        cout << "Error al abrir el fichero para escribir\n";
  }
  else
        cout << "Error al abrir el fichero para leer\n";
  return 0;
}
```

Operaciones en ficheros binarios (1/4)

- Antes de describir las operaciones de lectura y escritura en ficheros binarios recordemos que una característica importante de este tipo de ficheros es que podíamos acceder directamente a un dato sin tener que leer los anteriores.

- Para poder realizar este acceso directo, necesitaremos por tanto unas funciones específicas.

- Método de acceso directo
- El acceso directo consiste en poder mover la ventana del fichero a la posición del fichero que queramos, sin necesidad de leer todas las anteriores.
- En C++ la ventana del fichero corresponde únicamente a un carácter y se mueve carácter a carácter, por tanto la posición corresponderá al número de caracteres anteriores y NO al número de datos. La primera posición del fichero es siempre la posición 0.

- Ejemplo: Supongamos el siguiente fichero f con su contenido:

Fichero f
```
HOLA\n
PERE\n
```

- Carácter en la posición 0: H
- Carácter en la posición 3: A
- Carácter en la posición 6: P (en DOS o WINDOWS)

En el sistema operativo Windows, el salto de línea se escribe en fichero utilizando 2 caracteres, concretamente los correspondientes a los códigos 13 y 10.

Operaciones en ficheros binarios (2/4)

- Los métodos utilizados para el acceso directo son los siguientes:

- Para ifstream:
  nombre_fichero_logico.seekg(pos)
  nombre_fichero_logico.tellg()

- Para ofstream:
  nombre_fichero_logico.seekp(pos)
  nombre_fichero_logico.tellp()

- seekp(pos) o seekg(pos) coloca la ventana del fichero en la posición pos. tellg() o tellp() devuelven un entero que indica la posición actual de la ventana del fichero.

- **Ejemplo**:

```
// Colocar la ventana del fichero al principio del fichero
    f.seekg(0);
// Ir a la posición 6 (7º carácter) de f
    f.seekg(6);
    cout << f.get();          ->  P
    cout << f.tellg();        ->  7
```

El método tellg ha devuelto 7 porque al leer el carácter, la ventana se ha movido al siguiente carácter.

Operaciones en ficheros binarios (3/4)

- Lectura y escritura de ficheros binarios
- En C++ la lectura en binario se realiza mediante el método read y la escritura mediante el método write. En ambos casos hay que pasar como primer parámetro un puntero de tipo char a la variable a leer o escribir, y como segundo dato el número de bytes a leer o escribir.

- **Ejemplo: Lectura de enteros en binario**.

```
#include <fstream>
#include <iostream>
using namespace std;
int main()
{
    ifstream f;
    int dato;

    f.open("datos.bin");
    if(!f)
        cout << "Error abriendo el fichero" << endl;
    else
    {
        while(f.read((char *)(& dato), sizeof(dato) ) )
            cout << dato << endl;
        f.close();
    }
    return 0;
}
```

Operaciones en ficheros binarios (4/4)

- **Ejemplo: Escritura de 10 enteros en binario.**

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

int main()
{
    ofstream f;
    int i;

    f.open("datos.bin");

    if(!f)
        cout << "Error abriendo el fichero" << endl;
    else
    {
        for(i = 1; i <= 10; i++)
            f.write((char *)(& i), sizeof(i) );
        f.close();
    }
    return 0;
}
```