**Objetivos de la práctica:**
- Practicar uso de ficheros: abrir, cerrar y tratamiento de información contenida en el fichero.

## Uso de Ficheros

Todas las estructuras de datos vistas hasta ahora utilizan la memoria principal. Esto hace que los datos se pierdan cuando se termina un programa. Los ficheros son una estructura especial que utiliza la memoria secundaria. Es decir, se guardan en memoria no volátil y por ese motivo no se pierde su contenido cuando se apaga el programa.

Para cualquier científico este tipo de datos es de mucha importancia ya que una gran cantidad de medidas o pruebas se guardarán en ficheros que posteriormente serán analizados o procesados buscando conclusiones. En este sentido mientras anteriormente el resultado de nuestros ejercicios era un solo fichero (*.cpp), ahora tendremos además de nuestro fichero de programa uno o varios ficheros de datos (normalmente *.dat o *.txt).

### Tipos de acceso a la información y ficheros de texto

Para acceder a los datos disponibles en un fichero se suele disponer de 2 tipos de acceso: secuencial y directo.

- Acceso secuencial: el acceso a los datos del fichero sólo puede hacerse hacia delante a partir del primer elemento y siempre de uno en uno.
- Acceso directo: el acceso a los datos del fichero se puede hacer directamente en cualquier posición del fichero. Es un acceso similar al utilizado en los arrays usando un índice.

Nosotros solamente trabajaremos con acceso secuencial. Es más, solamente trabajaremos con ficheros de texto. Es decir ficheros que contienen una secuencia de caracteres separados por saltos de línea.

### Ficheros lógicos y ficheros físicos

En C++ los ficheros son un tipo más de datos, y un fichero concreto es referenciado a través de una variable de tipo fichero. Es lo que se suele llamar fichero lógico. En C++ existen 2 tipos de datos para declarar ficheros:

| | |
|---|---|
| ifstream | para declarar ficheros de entrada (de donde vamos a leer) |
| ofstream | para declarar ficheros de salida (donde vamos a escribir) |

Ejemplo de declaración de una variable tipo fichero de salida:

```
ofstream f;          //f variable tipo fichero de salida
```

Para utilizar estos tipos habrá que incluir una cabecera nueva de la siguiente forma:

```
#include <fstream>
```

La variable *f* anterior, para que sea de utilidad debe estar asociada a un fichero “real” (por ejemplo datos.txt). Este fichero es llamado fichero físico.

**Operaciones con ficheros**

Para relacionar los ficheros lógicos y físicos anteriores necesitamos una operación, que llamamos de apertura del fichero:

```
f.open ("datos.txt");   //ahora ya podemos usar el fichero
```

Así siempre que queramos hacer alguna operación con un fichero, primero lo abrimos, luego trabajamos con él y finalmente lo cerramos:

```
f.close();              //cierre del archivo f
```

Cuando se abre un archivo nos estamos posicionando sobre el primer elemento del archivo. Si el archivo que se ha abierto es para leer datos de él (ifstream), debe existir con anterioridad, o sino esta operación generará un error (no se puede abrir un fichero que no existe para leer su contenido).

Si el fichero es de salida (ofstream) se crea un fichero nuevo, y si ya existía su contenido queda borrado.

El modo de apertura de un archivo se puede modificar de la siguiente forma:

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

consiguiéndose abrir el fichero para añadir datos en él, de manera que no se borra su contenido y los datos que se añadan se pondrán a partir del último elemento. Si el fichero no existe al intentar abrirlo, genera un error.

Como ya se ha comentado hay varios casos en los que la tratar de abrir un archivo se puede generar un error. Para controlar este error, se deberá siempre utilizar estas sentencias:

```
if (!f )
    cout << "Error abriendo fichero" << endl;
```

En el caso de que no haya error (else) se pondrá el resto de nuestro programa.

Nota: Si queremos abrir un fichero cuyo nombre está almacenado en una variable de tipo string, para poder utilizar la operación de apertura de ficheros tendremos que hacer lo siguente:

```
string nombre = "datos.txt";
guia.open(nombre.c_str());
```

**Operaciones de entrada y salida de datos**

Son exactamente las mismas que se puede realizar con la entrada (cin) y salida (cout) estándar.

Es decir para poner el contenido de una variable x (de cualquier tipo simple) en un fichero:

```
f << x;
```

Mientras que para leer algo de un fichero y almacenarlo en un variable x:

```
    f >> x;
```

***Ejemplo***: Programa que escribe los números del 1 al 10 en el fichero datos.txt

```
#include<stdlib.h>
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
    ofstream f;    //fichero de salida
    int i;

    f.open("datos.txt");
    if(!f)
        cout << "Error abriendo el fichero" << endl;
    else
    {
        for(i = 1; i <= 10; i++)
           f << i << endl; //escribe el contenido de i y salta de línea

        f.close();
    }
    system (“pause”);
    return 0;
}
```

***Ejemplo***: programa para leer los 10 números enteros y mostrarlos por pantalla:

```
#include<stdlib.h>
#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();
    }
    system (“pause”);
    return 0;
}
```

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;
```

De esta forma, la instrucción anterior leerá, mientras sea posible, todos los datos del fichero, quedando el ejemplo de la siguiente forma:

```
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();
    }
    system (“pause”);
    return 0;
}
```

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.

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

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();
    }
    system (“pause”);
    return 0;
}
```

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.

## 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.

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

struct Telefono
{
    string nombre;
    string telefono;
};

bool F_IntroTel(ifstream & f, Telefono & tel);  //prototipo

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))
        {
            cout << tel.nombre << endl;
            cout << tel.telefono << endl;
            cout << endl;
        }
        guia.close();
    }
    system (“pause”);
    return 0;
}

/*************************************************************/
/*F_IntroTel     lee una estructura compuesta por un nombre y */
/*               un telefono                                 */
/*f     fichero del cual quiero leer                         */
/*tel   estructura donde voy a colocar lo leido              */
/*************************************************************/

bool F_IntroTel(ifstream & f, Telefono & tel)
{
    getline(f, tel.nombre);
    getline(f, tel.telefono);
    return (!f.eof());
}
```

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).

**Nota importante**: el fichero que se lee debe tener un último salto de línea después del último dato o no se leerá la última estructura.

**Resumen:**

**1.** No olvidar incluir la cabecera: `#include <fstream>`

**2.** Solamente se practicará con ficheros secuenciales de tipo texto:

- ✓ ficheros que guardan únicamente caracteres
- ✓ el acceso a la información es secuencial: para llegar a un lugar de fichero hay que pasar por todos los datos anteriores

**3.** Los ficheros se pueden abrir para leer su contenido o para escribir en ellos. Nunca leeremos y escribiremos simultáneamente (nota: *fichero* es solamente el nombre de la variable):

Tipo para definición de un objeto fichero de escritura: `ofstream fichero;`
Tipo para definición de un objeto fichero de lectura: `ifstream fichero;`
Apertura del fichero:

```
fichero.open (“nombre_fichero”);
```

Cierre del fichero:

```
fichero.close ();
```

Nota: por defecto el Dev C++ deja el fichero creado en el mismo directorio donde está el compilador:

```
fichero.open(“c:\\tmp\\hola.txt”); // deja en fichero hola.txt en el directorio
                                   // c:\tmp (notad la doble \)
```

**4.** Escritura y lectura. La lectura y escritura de información de se realiza a través los operadores << y >> (de forma equivalente al uso de cin y cout):

```
fichero << dato;  //donde dato es cualquier tipo de dato convertido a carácter. Escritura en
                  // fichero
fichero >> dato;  // lectura desde fichero
```

**5.** Fin de fichero. Para leer un fichero hasta su fin se empleará:

`while (fichero >> dato)` // esta condición proporcionará FALSE cuando se llegue al final.

**6.** Estructuras. No se puede leer ni almacenar una estructura directamente en un fichero, sino que debe procederse por separado con cada uno de sus campos:

ejemplo:
```
struct complejo{
        int real;
        int imag;
      }
```

Bien: `salidafich << complejo.real << “ “ << complejo.imag;`
Mal: salidafich << complejo;