TEMA 7: Ficheros

7.1.-Concepto de fichero

Todas las estructuras de datos que hemos visto hasta ahora utilizan memoria principal. Esto tiene dos limitaciones importantes:

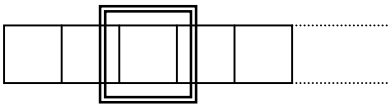
1. Los datos desaparecen cuando el programa termina.
2. La cantidad de los datos no puede ser muy grande debido a la limitación de la memoria principal.

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.

7.2.-Tipos de acceso

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.

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.

7.3.-Ficheros binarios y ficheros de texto

Existen dos tipos principales de ficheros:

- Ficheros de texto: Contienen secuencias de caracteres separadas por saltos de línea. El teclado (entrada estándar) y la pantalla (salida estándar) se consideran también ficheros de texto. Al leer o escribir variables de un fichero de texto se pueden realizar ciertas conversiones. Por ejemplo, cuando escribimos un entero con valor 10, este entero se convierte en los caracteres '1' y '0'.
- Ficheros binarios: Contienen secuencias de elementos de un tipo determinado de datos. Los elementos se almacenan en el fichero exactamente igual que están almacenados en memoria principal, es decir, al leer o escribir no se realiza ningún tipo de conversión. Un fichero binario es por tanto similar a un vector.

En este tema utilizaremos principalmente ficheros de texto. Los ficheros binarios los veremos en el apartado 7.7.

7.4.-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)
ofstream    // Para declarar ficheros de salida (out)
```

Para utilizar estos tipos hay que incluir antes el fichero de cabecera `<fstream.h>`.

Ejemplo:

```
ofstream f;
```

Esta sentencia nos declara una variable (fichero lógico) de tipo fichero de salida.

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*.

En C++ esta operación se realiza con la instrucción open:

```
nombre_fichero_logico.open (nombre_fichero_fisico);
```

Ejemplo:

```
f.open("datos.txt");
```

A partir de ese momento ya podemos utilizar el fichero.

7.5.-Procesamiento de un fichero

Siempre que queramos realizar cualquier operación con ficheros se debe seguir el siguiente esquema:

Apertura de Fichero → Operaciones → Cierre del fichero

7.5.1.-Apertura del fichero:

La **apertura** se realiza con la instrucción open, como acabamos de ver.
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);
```

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.

7.5.2.-Operaciones:

Las **operaciones** que se pueden realizar sobre un fichero 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.

Ejemplo:

Para escribir el numero 10 en el fichero f:

```
f << 10;
```

Para escribir un string:

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

7.5.3.-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 (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();
```

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

```
#include<fstream.h>

int main()
{
    ofstream f;
    int i;

    // APERTURA del fichero
    f.open("datos.txt");
    if(!f)
        cout << "Error abriendo el fichero" << endl;
    else
    {
        // OPERACIONES sobre el fichero
        for(i = 1; i <= 10; i++)
            f << i << endl;

        // CIERRE del fichero
        f.close();
    }
    return 0;
}
```

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

```
#include<fstream.h>
#include<iostream.h>

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;
}
```

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.

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

```
#include<fstream.h>
#include<iostream.h>
```

```
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;
}
```

Esta forma de leer del fichero se puede utilizar con cualquier tipo de lectura con >> y también con `getline`.

7.5.4.-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.

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

```
#include<fstream.h>
#include<iostream.h>

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;
}
```

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.

7.5.5.-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.

Ejemplo:

```
#include<fstream.h>
#include<iostream.h>

struct Telefono
{
    string nombre;
    string telefono;
};

bool F_IntroTel(ifstream & f, Telefono & tel)
void EscribeTel(Telefono tel);

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;
}

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

Por último conviene indicar que los ficheros se han de pasar siempre como parámetros por referencia, no importa si los vamos a modificar o no. En la función anterior `F_IntroTel` se puede ver un ejemplo.

7.6.-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:

Fichero f

```
HOLA\n
PEPE\n
```

Carácter en la posición 0: `H`

Carácter en la posición 3: `A`

Carácter en la posición 6: `P` (en DOS o WINDOWS)[1]

Métodos utilizados para el acceso directo:

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.

[1] 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.

7.7. Ficheros binarios

En los ficheros binarios, los datos se leen o escriben sin ningún tipo de transformación. Es un volcado directo de memoria a fichero o de fichero a memoria.

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 numero de bytes a leer o escribir.

Ejemplo: *Lectura de enteros en binario.*

```
#include<fstream.h>
#include<iostream.h>

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;
}
```

Ejemplo: *Escritura de 10 enteros en binario.*

```
#include<fstream.h>
#include<iostream.h>

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;
}
```