Ficheros en C++

Todos los programas vistos hasta ahora trabajaban con información almacenada en memoria principal, no obstante, hay situaciones en que esto no es apropiado. Algunas de esas situaciones podrían ser:

- Los datos con los que necesita trabajar el programa son demasiado grandes (ocupan mucha memoria) para que entren en la memoria principal.
- Nos interesa mantener la información después de cada ejecución, necesitamos utilizar datos procedentes de otros programas (editores, etc.), o generar datos para que puedan ser utilizados por otros programas.

En dichos casos se utilizarán ficheros para contener la información en memoria secundaria (disco duro, disquetes, etc.).

Definición de Fichero:
Es una colección de elementos lógicamente relacionados y almacenados en memoria secundaria. A más bajo nivel, un fichero es una secuencia de bits almacenado en algún dispositivo externo (como por ejemplo uno de memoria secundaria).

En C++ un fichero es simplemente un flujo externo que se puede abrir para entrada (dando lugar a un *flujo de archivo de entrada* que, para simplificar, llamaremos simplemente *archivo o fichero de entrada*), para salida (dando lugar a un *flujo de archivo de salida* que, para simplificar, llamaremos simplemente *archivo o fichero de salida*) o para entrada-salida (archivo o fichero de entrada-salida o archivo de E/S).

C++ soporta dos tipos de archivos: de texto y binarios. Los primeros almacenan datos como códigos ASCII. Los valores simples, tales como números y caracteres están separados por espacios o retornos de carro. Los segundos almacenan bits de forma directa (por lo que no se necesitan separadores) y se necesita usar la dirección de una posición de almacenamiento.

Una biblioteca en C++ que proporciona "funciones" y operadores para el manejo de ficheros es la biblioteca *fstream.*

```
#include <fstream>
```

Definición, Apertura y Cierre de Ficheros.

Declaración de Variables de tipo "Fichero":

```
ifstream descriptor;                    // Para ficheros de entrada
ofstream descriptor;                    // Para ficheros de salida
```

Apertura de Ficheros de Texto (supongamos TCadena nombre):

```
in.open(nombre);                       // Apertura para Lectura
out.open(nombre);                      /* Apertura para Escritura.
                                          (borra el contenido si lo hubiera) */
out.open(nombre,ios::app);             // Apertura para añadir datos al final
in.open(nombre,ios::in|ios:: out);  // Apertura para Lectura y Escritura

out.open(nombre,ios::in|ios:: out); // Apertura para Lectura y Escritura
```

Apertura de Ficheros Binarios (supongamos TCadena no

```
in.open(nombre,ios::binary);              // Apertura para Lectura
out.open(nombre,ios::binary);             /* Apertura para Escritura.
                                             (borra el contenido si lo hubiera) */
out.open(nombre,ios::binary| ios::app); // Apertura para añadir datos al final
in.open(nombre, ios::binary |             // Apertura para Lectura y Escritura
                ios::in | ios:: out);
out.open(nombre, ios::binary              // Apertura para Lectura y Escritura
         ios::in | ios:: out);
```

Ejemplo 1:

Abrir el fichero de texto "entrada.txt" como fichero de entrada (para lectura).

```
ifstream in;            // descriptor del fichero a abrir
in.open("entrada.txt"); // Apertura del fichero;
```

Ejemplo 2:

Crear el fichero de texto "salida.txt" como fichero de salida (para escritura, si existe borra su contenido)

```
ofstream out;            // descriptor del fichero a abrir
out.open("salida.txt"); // Apertura del fichero;
```

Ejemplo 3:

Abrir el fichero binario "entrada.bin" como fichero de entrada (para lectura).

```
ifstream in;            // descriptor del fichero a abrir
in.open("entrada.bin", ios::binary); // Apertura del fichero;
```

Ejemplo 4:

Crear el fichero binario "salida.bin" como fichero de salida (para escritura, si existe borra su contenido)

```
ofstream out;            // descriptor del fichero a abrir
out.open("salida.bin", ios::binary); // Apertura del fichero;
```

Cierre de ficheros.

Un fichero anteriormente abierto y con un descriptor asociado a él debe ser cerrado con el fin de liberar los recursos asociados a él de la siguiente forma:

```
descriptor.close()
```

Detección de fin de fichero y otras funciones.

- La función `eof()` que devuelve **`true`** si se ha alcanzado el final del fichero y falso en cualquier otro caso. REQUIERE LECTURA ADELANTADA: Para que la función `eof()` devuelva un valor de verdad (actualizado).
- La funcioón `fail()` devuelve **`true`** si existe un error en una operación de flujo asociada al fichero
- La función `bad()` devuelve **`true`** si el flujo está corrupto.
- La función `good()` que devuelve **`true`** si no existe un error en una operación de flujo y **`false`** en caso contrario.

Antes de empezar a leer o escribir en un fichero es siempre conveniente verificar que la operación de apertura se realizó con éxito.

Ejemplo 5:

```
ifstream in;

in.open("F1.dat");
if (in.fail() || in. bad())
{
  cout << "\n Incapaz de crear o abrir el fichero ";   // salida estándar
  cout << " para salida " << endl;
}
else
{
    // Se opera sobre el fichero
}
```

Ficheros de texto

La lectura y la escritura de datos en un archivo de texto es exactamente igual que la lectura/escritura por pantalla pero usando los descriptores de fichero en vez de *cin* y *cout*. Recuerde que el operador >> no consume el separador por lo tanto, al leer datos de un fichero de texto habrá que usar *descriptor.ignore()* después de cada lectura que no se haga con *getline(...)*.

Ejemplo 6:

El siguiente programa escribe tres líneas en un fichero llamado "EJEMPLO6.TXT" que se crea en el programa (si ya existe borramos su contenido). Cada línea consta de un entero, un real y una cadena de caracteres. Los datos en cada línea están separados por espacios en blanco.

```
#include <fstream>   // Biblioteca para el manejo de ficheros
#include <iostream>  // Biblioteca para la entrada-salida estándar
#include <cstdlib>   // Biblioteca estándar de C
using namespace std;

int main()
{
  ofstream fichout;

  fichout.open("EJEMPLO6.TXT");
  fichout << "Alumno 1" << " " << 5.0 << " " << "APROBADO" << endl;
  fichout << "Alumno 2" << " " << 1.1 << " " << "SUSPENSO" << endl;
  fichout << "Alumno 3" << " " << 8.0 << " " << "NOTABLE"  << endl;
  fichout.close();
}  // Fin del main
```

Ejemplo 7:

El siguiente programa lee el fichero de texto "EJEMPLO6.TXT", creado en el ejemplo anterior, y visualiza su contenido en el monitor.

```
#include <fstream>   // Biblioteca para el manejo de ficheros
#include <iostream>  // Biblioteca para la entrada-salida estándar
#include <cstdlib>   // Biblioteca estándar de C
using namespace std;

typedef char TCadena[30];

int main()
{
  ifstream fichin;   // declaracion del fichero
  int i;
  float r;
  TCadena cad;

 fichin.open("EJEMPLO5.TXT");
 if (fichin.fail() || fichin.bad())
 { cout << "Incapaz de crear o abrir el fichero " << endl;
 }
 else
 { fichin >> i;              // Observese la lectura adelantada!!!
   fichin.ignore();
   while (!fichin.eof())
   {
     fichin >> r;
     fichin.ignore();
     fichin.getline(cad,30,'\n');

     cout << i << " " << r << " " << cad << endl;
```

```
      fichin >> i;
      fichin.ignore();
     }
     fichin.close();
   }  // Fin del main
```

Ficheros Binarios

Los ficheros binarios tiene la información tal cual está en memoria, es decir, sin convertirla a texto, La manera de leer y escribir en ficheros binarios consiste en utilizar las funciones *read()* y *write()*.

```
descriptor.read((char *)(&c),  num);
```

```
Descriptor.write((char *)(&c), num);
```

donde `c` es una variable de cualquier tipo **PERO PASADA COMO (char *)** por referencia, y `num` es un entero o una variable de tipo entero, ejecuta una lectura/escritura (a partir de la posición actual del cursor del fichero asociado a la variable *descriptor*) de *num* bytes del fichero. Normalmente, para expresar el tamaño o número de bytes a leer/escribir se usa la función `sizeof` que nos retorna el número de bytes que ocupa una variable o tipo. Por ejemplo, para leer/ escribir un número entero en binario, usaremos:

```
int x;
...
descriptor.read ((char *)(&x),sizeof(int));
descriptor.write((char *)(&x),sizeof(int));
```

Ejercicio.-

Elaborar un programa que visusalice el siguiente menú, cuatro opciones adicionales que permitan la lectura de y escritura en fichero tanto de texto como binario. Para ello el menú debe quedar de la siguiente forma:

```
        MENU
        ====
Elaborado Por : Nombre Apellidos
Fecha:
  A. Insertar Persona en la Agenda.
  B. Borrar Persona de la Agenda.
  C. Imprimir Agenda.
  D. Leer la agenda de fichero de texto.
  E. Escribir la agenda en fichero de texto.
  F. Leer la agenda de fichero binario.
  G. Escribir la agenda en fichero binario.
  X. Salir del Programa

  Introduzca su opción:
```

Debemos tener en cuenta que las opciones relacionadas con ficheros siempre nos preguntarán por el nombre del fichero del que deseamos leer o escribir. Además, dado que el proceso de lectura implica la carga de una nueva agenda, las entradas que anteriormente existieran en la agenda contenida en memoria principal serán borradas.

El fichero de texto deberá contener por cada entrada de la agenda una línea con el nombre seguida de una segunda línea con el número de teléfono. Las entradas del fichero no están obligadas a seguir ningún orden.

```
Juan Sánchez Sánchez
555222201
José Antonio Pérez Lasa
555123321
María Pérez Pérez
555333333
```