LECTURA Y ESCRITURA DE FICHEROS EN C++

Por Sergio de la Cruz Rodríguez

El manejo de ficheros en C++ tiene características un tanto diversas al clásico manejo de ficheros que implementa el C estándar. La forma en que C++ maneja los ficheros es a través de flujos.

Como pudo verse en el semestre anterior, la entrada y salida de datos en C++ se realiza por medio de flujos (**cin** y **cout**), empleando los operadores de inserción (<<) y extracción (>>). La idea con respecto al trabajo con ficheros en C++ es emplear los ficheros como flujos de los que se extrae (lectura de un fichero) y se inserta (escritura de un fichero) información. Como veremos, para ello hace uso también del operador de inserción. La biblioteca que define estas operaciones es **fstream.h**. Es necesario entonces incluirla en el proyecto (**#include <fstream.h>**) para trabajar de esta forma los ficheros.

Existen tres tipos básicos de ficheros:

1. **Ficheros de lectura**: se abren para leer información de ellos.
2. **Ficheros de escritura**: se abren para escribir en ellos información. De existir ya el fichero, este se sobrescribe, perdiéndose los datos.
3. **Ficheros de escritura anexada**: semejantes a los de escritura, pero la información a escribir se anexa al final del fichero, evitando así que se pierdan los datos que existían previamente en el fichero.

La biblioteca fstream.h define tres clases que se corresponden con estos tres tipos de ficheros que mencionamos. Dichas clases son:

Clase	Tipo de fichero
fstream	Cualquiera de los tres mencionados.
ofstream	Ficheros de lectura.
ifstream	Ficheros de escritura

O sea, para acceder a un fichero, lo primero que hay que hacer es definir una instancia de la clase apropiada. Esta instancia entonces puede trabajarse como un flujo, tal y como se hace con **cin** y **cout**. En el caso de la clase **fstream**, al abrir el fichero es necesario especificar el modo de apertura. Los modos de apertura disponibles son:

Modo	Tipo de fichero
ios::out[1]	Ficheros de lectura.

[1] Se usa el operador de ámbito (::) porque estos indicadores de modo pertenecen a la clase ancestra **ios**, de la cual heredan no solo las tres clases, sino también los ya conocidos **cin** y **cout**.

ios::in	Ficheros de escritura
ios::append	Ficheros de escritura anexada

Como puede observarse, la clase **fstream** es la más general, pudiéndose trabajar con ella cualquier modo de apertura.

Definiendo el fichero

En C estándar para trabajar con ficheros debíamos crear un puntero de tipo FILE. Análogamente, en C++ se requiere crear una instancia de una de las tres clases anteriores para trabajar los ficheros como flujos. Esto sería:

```
fstream fich;      //Luego hay que especificar el modo de acceso
ifstream fich;     //Para leer de un fichero (modo lectura)
ofstream fich;     //Para escribir al fichero (modo escritura)
```

Para crear ficheros en modo de escritura anexada debe emplearse **fstream** y luego especificarle dicho modo.

Abriendo el fichero

Una vez creado el flujo, podemos proceder a abrir el fichero en cuestión. Para ello necesitamos, lógicamente, el nombre del fichero a abrir (o crear, en caso de que sea de escritura). Cada una de las tres clases tiene implementada la función miembro **open**, que en su forma más general toma como parámetros el nombre del fichero y el modo de apertura. En el caso de **ifstream** y **ofstream**, no es necesario especificar el modo, pues este es por defecto **ios::in** e **ios::out**, respectivamente. Veámoslo en algunos ejemplos:

```
ofstream fich; //Crea el flujo (fich)
fich.open("prueba.txt");//Crea el fichero prueba.txt para escribir
ifstream fil; //Crea el flujo (fil)
fil.open("prueba.txt");//Abre el fichero prueba.txt para leer
fstream apfich; //Crea el flujo (apfich)
apfich.open("prueba.txt",ios::append);
// Abre el fichero prueba.txt para anexarle información
```

Ahora bien, ¿cómo sabemos que el proceso de apertura/creación del fichero se realizó correctamente? Para ello puede emplearse la función miembro **is_open()**, que devuelve un valor distinto de cero (**true**) si el flujo está asociado a un fichero abierto. Veámoslo en un ejemplo:

```
ifstream fich("prueba.txt");
if (fich.is_open())
{/* Aquí el código para procesar el fichero */}
else {/* Mostrar un mensaje de error */}
```

En el código del ejemplo se creó el flujo y se abrió el fichero con una única instrucción, empleando el constructor de la clase. Como se ve, este constructor permite abrir/crear el fichero de una buena vez. Los parámetros que toma son los mismos que los de la función **open**.

Al igual que definimos el modo de apertura del fichero, podemos definir también el tipo de fichero. El tipo por defecto es ASCII. Si queremos especificar el tipo como binario, tenemos que añadir el especificador **ios::binary**. Si quisiéramos abrir un fichero binario nombrado **prueba.bin**, serviría la siguiente instrucción: `fstream fich("prueba.bin", ios::in | ios::binary)`. Como puede verse, se combinan ambos especificadores con un OR (|). De todas formas, la mayor parte de los ficheros que se usan son tipo ASCII, por lo que este es el tipo por defecto.

Otra función miembro muy útil es **eof()**, que devuelve que devuelve un valor distinto de cero (**true**) si se ha alcanzado el fin del fichero.

Cerrando el fichero

Cerrar el fichero previamente abierto al terminar de trabajar con él es tan sencillo como llamar a la función miembro **close()**. En nuestro caso, pudiera ser:

```
fich.close();
```

Leyendo y escribiendo en el fichero

Para leer y escribir en el fichero pueden emplearse las funciones miembros que se describen a continuación:

`ostream& put(char c)`	Escribe un carácter en el flujo de salida
`ostream& write(const char* s, int n)`	Escribe ***n*** bytes de la cadena ***s*** en el flujo de salida. Puede utilizarse para salida binaria
`istream& get(char& c)`	Lee un carácter del flujo de entrada y lo devuelve en el argumento pasado por referencia
`istream& get(char* s, int n, char c='\n')`	Introduce en ***s*** a lo más ***n*** caracteres del flujo de entrada (incluyendo el '\0') o hasta que encuentra el carácter de terminación (por defecto '\n'), o el fin de fichero. No retira el carácter de terminación del flujo de entrada.
`istream& getline(char* s, int n, char c='\n')`	Lee a lo más ***n-1*** caracteres del flujo de entrada o hasta que encuentra el

	carácter de terminación (por defecto '\n') o hasta el fin de fichero. Retira el carácter de terminación del flujo de entrada, pero no lo almacena.
istream& read(char* s, int n)	Lee ***n*** bytes del flujo de entrada y los deposita en la cadena ***s***. Se utiliza para entrada binaria.
istream& ignore(int n=1, int delim=EOF)	Ignora o descarta los ***n*** caracteres siguientes del flujo de entrada, o hasta que encuentra el carácter de terminación (por defecto el fin de fichero EOF).
istream& putback(char c)	Devuelve el carácter ***c*** al flujo de entrada.
int peek()	Lee un carácter del flujo de entrada pero sin retirarlo de dicho flujo; lo devuelve como valor de retorno.

Además, en el caso de la escritura de un fichero puede emplearse el operador de inserción (<<), tal y como se usa con **cout**. Veamos algunos ejemplos:

```
char frase[81];
fstream fi;
fi.open("datos.txt", ios::in);
while(fi.getline(frase, 80) != NULL)
m_lista.AddString(frase);// Añade la línea al listbox m_lista
fi.close();
```

Como se ve en el ejemplo, la referencia al flujo que devuelven estas funciones puede ser empleada para determinar si se ha alcanzado el fin del fichero (instrucción **while** en el ejemplo).

```
fi.getline(frase,80,' '); //Lee hasta el primer espacio en blanco
```

Otro ejemplo, ahora para la escritura en un fichero. El siguiente código genera 10 números aleatorios entre 0 y 100 y los guarda en líneas independientes del fichero numeros.txt.

```
#include <fstream.h>
ofstream fich("numeros.txt"); //Crea el fichero
if (!fich.is_open()) exit(1); //Si no pudo, salir del programa
for (int i = 0; i<10; i++) //Se entra al ciclo 10 veces
 fich << rand()%101 << endl;
```

```
fich.close();
```

La función **rand()** genera números aleatorios entre cero y un número bastante grande (32767). Para lograr que los números aleatorios estén dentro del rango previsto se toma el módulo de la división por 101, que lógicamente será un número entre 0 y 100.

Otro aspecto interesante es que los manipuladores de entrada/salida que se empleaban con **cout** también están disponibles. Los más usados son:

dec, hex y oct	establecen base para enteros
ws	se saltan los blancos iniciales
endl	se imprime un '\n' y se vacía el buffer de salida
flush	se vacía el buffer de salida
setw(int w)	establece la anchura mínima de campo
setprecision(int p)	establece el número de cifras
setfill(char ch)	establece el caracter de relleno

Por ejemplo, la instrucción

```
fich << hex << 255 << endl;
```

es equivalente a:

```
fich << hex <<"255\n";
```

Ambas escriben el número 255 en hexadecimal (FF) en el fichero y luego insertan un salto de línea.

Por último, aquí va otro ejemplo. Este código permite extraer números separados por comas (,) de un fichero. Pueden haber tantos espacios en blanco como se quiera entre los números. No es necesario que el último número de una línea termine con coma. Puden haber tantas líneas en el fichero como se quiera. Las líneas deben tener menos de 255 caracteres.

```
 char linea[250]; CString lin,cad; int c = 0;
 ifstream fich("numeros.txt");
 if (!fich.is_open())
{
       MessageBox("Fichero no abierto");
       exit(1);
 }
 while (!fich.eof())
 {
       fich.getline(linea,250); // Lee una línea
       lin = linea;     // La asigna a lin que es un CString
       lin.Remove(' '); // Borra los espacios en blanco de la
                        //cadena
       c = 0;
       while (1) // Ciclo infinito (se sale con break)
       {     if (lin.IsEmpty()) // Si la línea está vacía
                   break;       // no la procesa
             c=lin.Find(",");   // En c, la posición de la coma en
```

```
                                    // la cadena

        if (c == -1)  // Si no hay comas, c Find devuelve -1
                {
                        m_lista.AddString(lin); // Inserta el último
                                                //fragmento
                        break; // Pasa a procesar la próxima línea
                }

                cad = lin.Left(c); // Toma todos los caracteres hasta
                                   // la coma
                m_lista.AddString(cad); // Los inserta en el listbox
                lin = lin.Right(lin.GetLength()-c-1);
                // Elimina la subcadena extraída
         }
      }
      fich.close();
```

El código anterior echa mano de diversas funciones miembro de la clase **CString**. Pero ya eso es tema de otro artículo.

Bibliografía

1. Aprenda C++ como si estuviera en primero. Capítulo 6.