Archivos y su procesamiento

Los archivos digitales y más concretamente las bases de datos en las organizaciones, adquieren especial relevancia, ya que son utilizadas para almacenar los datos que resultan de las transacciones diarias. Las diversas herramientas de software actuales permiten la creación y administración de bases de datos que registran cientos de miles o millones de transacciones.

Algunas herramientas para la administración de bases de datos que los alumnos tendrán la oportunidad de conocer en el transcurso de sus estudios, son: MySQL, PostgreSQL, SQL Server, Oracle, SQLite.

C++, al igual que los demás lenguajes de programación, permite al programador el manejar archivos en disco por medio de una serie de comandos predefinidos de la biblioteca *fstream*. Para utilizar las funciones y métodos de esta biblioteca en un programa, debe incluirse por medio de una directiva, al igual que ya sabemos hacerlo con otras que hemos aprendido:

```
#include <fstream>
```

Básicamente existen dos tipos de archivos que podemos manipular: los archivos de texto y los archivos binarios. Los archivos de texto son todos aquellos cuyo contenido puede ser leído e interpretado directamente por las personas. Pero en una computadora no solo existen este tipo de archivos sino que por ejemplo, los archivos de sonido, de imágenes, de videos, hojas electrónicas, documentos elaborados en editores de documentos con formato, bases de datos de transacciones de una organización, mapas geográficos, etc., están almacenados en lo que se conoce como formato binario.

Aprenderemos en un primer momento a manipular archivos de texto y utilizaremos para ello algunos comandos que ya conocemos. Luego aprenderemos a manejar archivos en formato binario.

Manejo de los datos por medio de flujos

En C++ se maneja el concepto de flujo (*stream*). Esto se refiere a la traslación de un bloque de bytes, entre el programa y el punto de almacenamiento. El flujo puede ir del archivo al programa (flujo de entrada) o del programa hacia el archivo (flujo de salida), e incluso en ambos sentidos (flujo de entrada y salida).

Toda entrada o salida que se realiza en un programa de computadora se lleva a cabo a través de un flujo, que se conecta con un dispositivo o con un archivo de datos almacenado en un dispositivo. El flujo se maneja a través de un *buffer*, un espacio de memoria que servirá de intermediario entre el programa y el archivo que se encuentra en el dispositivo de almacenamiento de datos. Es decir, el programa no interactúa directamente con el archivo, sino que utiliza este espacio de memoria que es declarado dentro del programa y conectado con el archivo. De esta manera, el archivo tiene una representación lógica dentro del programa, tal y como la tienen el resto de variables.

Si el flujo es de entrada de datos, las operaciones de lectura realizan peticiones al sistema operativo para que traslade datos del archivo hacia el buffer y luego el programa los toma del buffer para procesarlos.

Si el flujo es de salida de datos, el programa deposita los datos en el buffer y luego le solicita al sistema operativo que estos sean trasladados desde el buffer hacia el archivo.

Si el buffer es de entrada y salida, servirá como zona de intercambio para realizar traslado de datos en ambos sentidos.

Los flujos, o buffers, son manipulados por medio de punteros: el puntero de lectura si el archivo se abre en esta modalidad; el puntero de escritura, si el archivo se abre en esta modalidad; o dos punteros, uno para lectura y otro para escritura, si el archivo se abre con ambas modalidades. Su funcionamiento es muy similar a como se hace en el buffer de teclado, del cual ya se ha hablado anteriormente. El lenguaje C++ nos permite tener un cierto dominio de estos punteros. Esto lo podremos ver posteriormente.

Apertura de archivos y sus modalidades

El primer paso para conectarse a un archivo mediante un flujo es ejecutar la apertura del archivo, que lleva implícita la creación del flujo.

Para abrir el archivo para salida se ejecutan las instrucciones:

ofstream Nombre lógico;

Nombre lógico.open(Nombre físico, Modalidad(es) de apertura);

O bien de esta forma:

ofstream Nombre lógico(Nombre físico, Modalidad(es) de apertura);

Donde:

Nombre lógico es un puntero a carácter que apunta al buffer que se crea en el proceso.

Nombre físico es el nombre del archivo en el dispositivo de almacenamiento, incluye la ruta de directorios.

Modalidad(es) de apertura pueden ser varias modalidades, pueden colocarse varias al mismo tiempo, separadas por |. Si no se proporciona este argumento se asume la modalidad de salida (output). Si se especifica, se puede escribir de esta forma: ios::out.

Para abrir un archivo en modo de salida, es decir, para grabar datos en él, debe tenerse el cuidado de no sobrescribir la información que ya contiene si no se desea perderla. Esto es, la apertura puede ser destructiva, pero es algo que se puede controlar.

Para abrir el archivo para entrada se ejecutan las instrucciones:

ifstream Nombre lógico;

Nombre lógico.open(Nombre físico, Modalidad(es) de apertura);

O bien de esta forma:

ifstream Nombre lógico(Nombre físico, Modalidad(es) de apertura);

Donde:

Nombre lógico es un puntero a carácter que apunta al buffer que se crea en el proceso.

Nombre físico es el nombre del archivo en el dispositivo de almacenamiento, incluye la ruta de directorios.

Modalidad(es) de apertura pueden ser varias modalidades, pueden colocarse varias al mismo tiempo, separadas por |. Si no se proporciona este argumento se asume la modalidad de entrada (input). Si se especifica, se puede escribir de esta forma: ios::in.

Para abrir el archivo para entrada y salida se ejecutan las instrucciones:

fstream Nombre lógico;

Nombre lógico.open(Nombre físico, Modalidad(es) de apertura);

O bien de esta forma:

fstream Nombre lógico(Nombre físico, Modalidad(es) de apertura);

Donde:

Nombre lógico es un puntero a carácter que apunta al buffer que se crea en el proceso.

Nombre físico es el nombre del archivo en el dispositivo de almacenamiento, incluye la ruta de directorios.

Modalidad(es) de apertura pueden ser varias modalidades, pueden colocarse varias al mismo tiempo, separadas por |. Si no se proporciona este argumento se asumen las modalidades de entrada (input) y salida(*output*). Si se especifican, se pueden escribir de esta forma: *ios::in | ios::out*.

Las modalidades de apertura pueden utilizarse en todas las formas de apertura: input, *in*; output, *out*; binary, *binary*; at end, *ate*; append, *app*; truncate, *trunc*.

Cierre de los archivos

El último paso después de haber realizado el procesamiento de un archivo es ejecutar el cierre del mismo antes de finalizar el programa. Esto se realiza por medio del comando *close*, su sintaxis es la siguiente:

Nombre lógico.close( );

Debe realizarse un *close* por cada archivo que se haya abierto en el programa. Aunque, en teoría, el sistema operativo debe cerrar los archivos una vez el programa ha finalizado, es buena práctica ejecutar esta instrucción siempre.

Ejemplos:

1) Realizar la apertura y cierre de un archivo que se llame *miPrimerArchivo.txt*, el cual no existe en su disco. Debe abrirse en modo de salida. Ejecute el programa y luego revise la carpeta donde se encuentra el archivo ejecutable para verificar que el archivo ha sido creado.

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

using namespace std;

int main(void)
{
      ofstream archivo;
      archivo.open("miPrimerArchivo.txt");

     archivo.close( );

     return 0;
}
```

Para resolver este problema el archivo se debe abrir en modo de salida con ofstream. Pues si se abre para entrada, C++ esperaría ya encontrarlo en el disco. La extensión del archivo no importa, pero se ha sugerido que sea .txt para que se pueda abrir automáticamente con Bloc de Notas y notar que el archivo está vacío. La siguiente instrucción cierra el archivo pues no vamos a realizar ninguna operación con él.

Vaya con el explorador de Windows a la carpeta del proyecto y verá que allí se encuentra el archivo miPrimerArchivo.txt. Revise la fecha y hora de creación y su tamaño en bytes. Pulse dos veces el archivo para que se abra con Bloc de Notas y verifique que no tiene ningún contenido.

Dado que el archivo no existía antes de ejecutar el programa, la primera instrucción del programa, que es la operación de apertura implicó la creación del mismo. El sistema operativo buscó un sector disponible del disco duro, reservó un espacio en la tabla de localización de archivos del disco y lo registró. Luego, el programa ejecutó la segunda instrucción del programa, que es el cierre del archivo. El sistema operativo colocó la marca de fin de archivo (Ctrl-Z) en el sector reservado para el archivo y realizó la desconexión entre el flujo y el programa, y entre el flujo y el archivo físico.

2) Realizar la apertura de un archivo que se llame *miArchivo.txt*, el cual no existe en la carpeta del proyecto de este programa. Debe abrirse en modo de entrada. Verifique con una instrucción *if* si la apertura es o no exitosa.

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

using namespace std;

int main()
{
      ifstream archivo;
      archivo.open("elArchivo.txt");
```

```
    if(!archivo)
        cout << "El archivo no existe" << endl;
    else
        cout << "El archivo si existe" << endl;

    archivo.close( );

    return 0;
}
```

Grabar en archivos de texto

Para almacenar un texto en un archivo podemos utilizar:

a) El operador de redirección de flujo hacia la salida: **<<**. Se utiliza como ya sabemos hacerlo, con la diferencia que en esta ocasión ponemos a su izquierda el nombre del flujo.

b) El comando put( ). Coloca un carácter en el archivo de salida. Su sintaxis es: Nombre lógico.put(Carácter). Su argumento puede ser, por ejemplo, un carácter entre comillas simples o una variable de carácter.

c) El comando write( ). Copia un conjunto de carácteres en el archivo de salida. Su sintaxis es: Nombre lógico.write(Bloque a copiar, Tamaño). Donde Bloque a copiar es la cadena entre comillas o la variable que contiene la secuencia de carácteres a copiar. El Tamaño es la longitud de esta cadena, que puede ser calculado con *sizeof( )* o de cualquier otra forma, debe ser un número entero.

Por ejemplo:

1) Escriba un programa que grabe una frase en un archivo de texto. Luego vaya a la carpeta del proyecto y verifique el contenido del archivo abriéndolo con Bloc de notas.

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

using namespace std;

int main() {
    ofstream archivo("elArchivo.txt");

    archivo << "Me llamo Anita" << endl << endl;

    archivo.close( );

    return 0;
}
```

Vaya a la carpeta del proyecto y abra el archivo elArchivo.txt con Bloc de Notas. Observará que contiene la frase *Me llamo Anita* y dos cambios de líneas.

IMPORTANTE 1: Dependiendo del sistema operativo y de la versión del compilador y su origen el endl podría no funcionar. En ese caso, debería utilizarse el carácter **\n**.

IMPORTANTE 2: En algunos sistemas operativos, aunque se grabe el **\n**, no se verá el cambio de línea cuando se cargue el archivo con un editor de texto. Es decir, el **\n** está allí, pero el editor de texto no lo refleja colocando la siguiente línea abajo. En ese caso debe incluir la secuencia **\r\n** (Retorno de carro - Cambio de línea).

Suele haber diferencia entre compiladores de Linux y compiladores de Windows. También, si un compilador fue creado inicialmente para un sistema operativo y luego surgen de ese compilador versiones para otro sistema operativo, podría aun conservar su requisito original. Todo ello usted lo debe considerar cuando esté haciendo las pruebas a sus programas.

2) Escriba un programa que grabe varias líneas en un archivo de texto. Luego vaya a la carpeta del proyecto y verifique el contenido del archivo abriéndolo con Bloc de Notas.

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

using namespace std;

int main() {
    ofstream archivo("archivo03.txt");

    archivo << "Me llamo Anita,\n";
    archivo << "¿y vos?\n";
    archivo << "Yo me llamo Pepito\n";
    archivo << "Y soy bien portado\n";

    archivo.close( );

    return 0;
}
```

Debe tomarse en cuenta que el comando *ofstream*, tal y como se ha utilizado en los ejemplos anteriores, es destructivo. Es decir, la apertura implica la creación del archivo y colocación del puntero del archivo al inicio, de tal manera que las salidas de una nueva ejecución del programa destruyen el contenido anteriormente almacenado. Para abrir un archivo de salida y añadir contenido hay que utilizar el modo *ios::app*, que coloca el puntero de escritura del archivo al final luego de la apertura.

Por ejemplo:

1) Escriba un programa que grabe una línea en un archivo de texto. Luego vaya a la carpeta del proyecto y verifique el contenido del archivo abriéndolo con Bloc de Notas. Luego cierre el Bloc de Notas. Cambie el contenido de la frase a imprimir, salve, compile y realice una nueva corrida. Luego diríjase nuevamente a la carpeta del proyecto y revise de nuevo el contenido del archivo con Bloc de Notas.

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

using namespace std;

int main() {
    ofstream archivo("archivo05.txt", ios::app);
```

```
    archivo << "Una frase\n";

    archivo.close( );

    return 0;
}
```

De igual forma puede ejecutar un ciclo de escritura implementando una instrucción iterativa.

Por ejemplo:

1) Implemente una escritura cíclica con datos provenientes desde teclado, hasta que el usuario ya no desee continuar con el proceso.

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

using namespace std;

int main() {
    string frase;
    cout << "LLENAR UN ARCHIVO CON DATOS DESDE TECLADO" << endl << endl;

    ofstream archivo;
    archivo.open("archivo06.txt");

    cout << "Digite una frase o Ctrl-Z para finalizar:" << endl;
    while(getline(cin, frase, '\n')){
        archivo << frase << endl;
        cout << "Digite una frase o Ctrl-Z para finalizar:" << endl;
    }

    archivo.close( );

    return 0;
}
```

Lectura de archivos de texto y manejo de archivos en modo binario los veremos en otro documento.