Manejo de archivos en C++

C++ posee 3 clases que nos permiten realizar acciones de entrada y salida desde/hacia archivos:

- `ofstream`: Clase que permite escribir en archivos.
- `ifstream`: Clase que permite leer desde archivos.
- `fstream`: Clase que permite leer y escribir en archivos.

Para utilizar estas clases es necesario incluir la directiva `fstream`:

```
#include <fstream>
```

Ejemplo básico:

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

using namespace std;

int main () {
  // Declara la variable y abre el archivo
  // para escribir datos
  ofstream archivo;
  archivo.open("ejemplo.txt");

  // En vez de cout, usamos la variable archivo
  // para insertar los datos al stream
  archivo << "¡Hola Mundo!" << endl;

  // Cierra el archivo
  archivo.close();

  return 0;
}
```

Para abrir un archivo se utiliza la función `open(filename, mode)`, donde `filename` es la ruta, *relativa o absoluta*, al archivo (incluyendo su extensión) y `mode` es un parámetro ***opcional*** con una combinación de las siguientes opciones o *flags*:

Flag	Descripción
`ios::in`	Abre para operaciones de entrada (lectura).
`ios::out`	Abre para operaciones de salida (escritura).
`ios::binary`	Abre en modo binario.
`ios::ate`	Asigna la posición del cursor al final del archivo. Si no se proporciona, la posición inicial será el comienzo del archivo.
`ios::app`	Todas las operaciones de escritura se realizan al final del archivo, agregando el nuevo contenido al ya existente.

`ios::trunc`	Si el archivo es abierto para operaciones de escritura, todo el contenido existente será eliminado y reemplazado por el nuevo contenido.

Todas estas opciones o *flags* pueden combinarse utilizando el operador *bitwise* OR: `|`.

```
ofstream archivo;
archivo.open("ejemplo.txt", ios::out | ios::app | ios::binary);
```

La función `open` de cada una de las clases `ifstream`, `ofstream` y `fstream` tienen un modo por defecto que es utilizado al abrir un archivo:

Clase	Descripción
`ofstream`	`ios::out`
`ifstream`	`ios::in`
`fstream`	`ios::in | ios::out`

Una forma alternativa de abrir un archivo, en una sola línea, es la siguiente:

```
fstream archivo("ejemplo.txt");
```

Para verificar si un archivo está abierto se utiliza la función `is_open()`:

```
if(archivo.is_open()) {
  // El archivo está abierto, proceder con las operaciones de lectura/escritura
}
```

Para cerrar un archivo utilizamos la función `close()`:

```
archivo.close();
```

Lectura y escritura de archivos planos

```
/*
 * Escritura de datos a un archivo plano
 */
#include <iostream>
#include <fstream>

using namespace std;

int main () {
  // Abre el archivo para escritura
  ofstream archivo("ejemplo.txt");

  // El archivo está abierto?
  if (archivo.is_open()) {
    // Envía los datos al stream de salida "archivo"
    archivo << "Esta es una línea de ejemplo" << endl;
    archivo << "Esta es otra línea...\n";
    // Cierra el archivo
    archivo.close();
  }
  // En caso de error alerta al usuario
  else cout << "Imposible abrir el archivo.";

  return 0;
}
```

```
/*
 * Lectura de un archivo plano
 */
#include <iostream>
#include <fstream>

// Directiva para utilizar strings
#include <string>

using namespace std;

int main () {
  string linea;
  ifstream archivo("ejemplo.txt");

  // El archivo está abierto?
  if(archivo.is_open()) {
    // Mientras existan líneas para leer
    while(getline(archivo, linea)) {
      cout << linea << '\n';
    }
    // Cierra el archivo
    archivo.close();
  }
  // En caso de error alerta al usuario
  else cout << "Imposible abrir el archivo.";

  return 0;
}
```

Verificación de estados

Las siguientes funciones verifican estados específicos del stream. Todas estas funciones retornan un valor del tipo `boolean`.

- `bad()` Retorna `true` si una operación de lectura o escritura falla. Por ejemplo, si intentamos escribir en un archivo que no ha sido abierto para escritura o no hay espacio suficiente.
- `fail()` Retorna `true` en los mismos casos que `bad`, pero también en el caso en que un error de formato ocurra, como cuando se intenta leer un entero pero se encuentra o lee un caracter.
- `eof()` Retorna `true` si un archivo abierto para lectura ha llegado a su fin.
- `good()` Es el *flag* más genérico. Retorna `false` en los mismos casos en que las funciones anteriores retornan `true`. Es importante señalar que `good` y `bad` no son opuestos exactos, ya que `good` verifica más estados a la vez.
- `clear()` Restablece el *flag* de estado.

Posicionamiento de streams

Todos los objetos *stream* mantienen internamente *al menos* una posición de lectura/escritura hacia la próxima operación:

- `ifstream` mantiene una posición interna de lectura (*get*) del elemento a ser leído en la próxima operación de entrada.
- `ofstream` mantiene una posición interna de escritura (*put*) del elemento a ser escrito en la próxima operación de salida.
- Finalmente, `fstream`, mantiene ambas posiciones.

`tellg()` y `tellp()`

Estas 2 funciones, sin parámetros, retornan la posición interna del elemento a ser leído (`tellg`) o escrito (`tellp`).

`seekg()` y `seekp()`

Estas funciones permiten cambiar la posición de lectura o escritura de la próxima operación del stream.

```
// Cambia la próxima posición de lectura a "position", contando desde el inicio
seekg(position);
// Cambia la próxima posición de escritura a "position", contando desde el inicio
seekp(position);
```

Es posible también especificar la cantidad a desplazamiento (*offset*) a partir de cierta dirección

```
// Cambia la próxima posición de lectura, desplazándose "offset" veces a partir de
"direction"
seekg(offset, direction);
// Cambia la próxima posición de escritura, desplazándose "offset" veces a partir de
"direction"
seekp(offset, direction);
```

Los valores posibles para `direction` son:

Flag	Descripción
`ios::beg`	offset contado desde el inicio del stream
`ios::cur`	offset contado desde la posición actual
`ios::end`	offset contado desde el inicio del stream

Ejemplo:

```
/*
 * Calcula el tamaño (en bytes) de un archivo
 */
#include <iostream>
#include <fstream>

using namespace std;

int main () {
  streampos inicio, fin;

  ifstream archivo("ejemplo.bin", ios::binary);

  inicio = archivo.tellg();
  archivo.seekg (0, ios::end);
  fin = archivo.tellg();
  archivo.close();

  cout << "El tamaño del archivo es: " << (fin - inicio) << " bytes.\n";

  return 0;
}
```

Nótese que en el ejemplo se ha utilizado el tipo de dato `streampos` para las variables `inicio` y `fin`.

`streampos` es un tipo de dato específicamente utilizado para buffers y posicionamiento de archivos. Es el tipo retornado por la función `tellg`. Los valores de este tipo pueden ser convertidos a un tipo entero lo suficientemente grande para contener el tamaño de un archivo. Soporta operaciones aritméticas.

Archivos binarios

Cuando tratamos con archivos binarios, la lectura y escritura de datos utilizando los operadores `<<` y `>>` o funciones como `getline` no son eficientes, ya que los datos no necesitan ser formateados ni tampoco serán organizados en líneas.

Los streams de archivos incluyen dos funciones específicamente diseñadas para leer y escribir **datos secuenciales**: `write` y `read`. Sus prototipos son:

```
read(memory_block, size);
write(memory_block, size);
```

- `memory_block` es del tipo `char*` (puntero a char), y representa la dirección de un array de bytes donde los elementos de datos de lectura son almacenados o desde dónde los elementos de datos a ser escritos son tomados.
- El parámetro `size` es un valor entero que especifica el número de caracteres a ser leídos o escritos desde/hacia el bloque de memoria.

```
/*
 * Lectura completa de un archivo binario
 */
#include <iostream>
#include <fstream>

using namespace std;

int main () {
  streampos tamano;
  char* bloque;

  // Abre el archivo para lecutra, en modo binario y
  // posiciona el cursor al final del archivo (para obtener su tamaño)
  ifstream archivo("ejemplo.bin", ios::in | ios::binary | ios::ate);

  // El archivo está abierto?
  if (archivo.is_open()) {
    // Obtiene el tamaño (en bytes) del archivo
    tamano = archivo.tellg();

    // Asigna la cantidad de bytes en base al tamaño del archivo
    bloque = new char[tamano];
    // Reposiciona el stream de lectura al inicio del archivo
    archivo.seekg(0, ios::beg);
    // Lee el total de datos y los almacena en "bloque"
    archivo.read(bloque, tamano);

    // Cierra el archivo
    archivo.close();

    cout << "El contenido se encuentra en memoria." << endl;

    // A partir de este punto se puede manipular el contenido en "bloque"
  }
  // En caso de error alerta al usuario
  else cout << "Imposible abrir el archivo.";

  return 0;
}
```