Archivos en C++

Vamos a ver cómo manejar archivos en C++ tanto en formato binario como en formato de texto. Cubriremos la creación, modificación, y eliminación de archivos.

Archivos de Texto

Creación y Escritura en Archivos de Texto

Para trabajar con archivos de texto en C++, utilizamos las clases std::ofstream para escritura y std::ifstream para lectura, que forman parte de la biblioteca <fstream>.

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

int main() {
	std::ofstream outFile("archivo.txt"); // Crear un archivo de texto para escritura
	if (!outFile) { // Verificar si el archivo se creó correctamente
		std::cerr << "No se pudo abrir el archivo para escritura." << std::endl;
		return 1;
	}

	outFile << "Hola, mundo!" << std::endl; // Escribir en el archivo
	outFile.close(); // Cerrar el archivo
	return 0;
}
```

Lectura de Archivos de Texto

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

int main() {
	std::ifstream inFile("archivo.txt"); // Abrir un archivo de texto para lectura
	if (!inFile) { // Verificar si el archivo se abrió correctamente
		std::cerr << "No se pudo abrir el archivo para lectura." << std::endl;
		return 1;
	}

	std::string line;
	while (std::getline(inFile, line)) { // Leer línea por línea
		std::cout << line << std::endl;
	}
	inFile.close(); // Cerrar el archivo
	return 0;
}
```

Modificación de Archivos de Texto

Para modificar un archivo de texto, puedes abrirlo en modo de lectura y escritura. Aquí hay un ejemplo de cómo hacerlo:

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

int main() {
	// Abrir el archivo en modo de lectura y escritura
	std::fstream file("archivo.txt", std::ios::in | std::ios::out);
	if (!file) {
		std::cerr << "No se pudo abrir el archivo." << std::endl;
		return 1;
	}

	std::string line;
	while (std::getline(file, line)) { // Leer todo el contenido
		std::cout << line << std::endl;
	}

	// Posicionar el puntero al final del archivo
	file.clear(); // Limpiar el estado de error
```

```
    file.seekp(0, std::ios::end); // Mover el puntero al final del archivo

    file << "Nueva línea añadida al final." << std::endl; // Escribir nueva línea
    file.close(); // Cerrar el archivo
    return 0;
}
```

Borrado de Archivos de Texto
Para eliminar un archivo, puedes usar la función remove de la biblioteca estándar:

```
#include <iostream>
#include <cstdio> // Para std::remove

int main() {
    if (std::remove("archivo.txt") != 0) { // Intentar eliminar el archivo
        std::cerr << "No se pudo eliminar el archivo." << std::endl;
    } else {
        std::cout << "Archivo eliminado exitosamente." << std::endl;
    }
    return 0;
}
```

Archivos Binarios

Archivos Binarios
Creación y Escritura en Archivos Binarios
Para trabajar con archivos binarios en C++, usamos las mismas clases std::ofstream y std::ifstream, pero con el modo binario.

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

struct Datos {
    int id;
    float valor;
};

int main() {
    std::ofstream outFile("archivo.bin", std::ios::binary); // Crear un archivo binario
    if (!outFile) {
        std::cerr << "No se pudo abrir el archivo para escritura." << std::endl;
        return 1;
    }

    Datos d = {1, 23.5};
    outFile.write(reinterpret_cast<const char*>(&d), sizeof(d)); // Escribir estructura
en el archivo
    outFile.close(); // Cerrar el archivo
    return 0;
}
```

Lectura de Archivos Binarios

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

struct Datos {
    int id;
    float valor;
};

int main() {
    std::ifstream inFile("archivo.bin", std::ios::binary); // Abrir archivo binario para
lectura
```

```
    if (!inFile) {
        std::cerr << "No se pudo abrir el archivo para lectura." << std::endl;
        return 1;
    }

    Datos d;
    inFile.read(reinterpret_cast<char*>(&d), sizeof(d)); // Leer datos del archivo
    std::cout << "ID: " << d.id << ", Valor: " << d.valor << std::endl;
    inFile.close(); // Cerrar el archivo
    return 0;
}
```

Modificación de Archivos Binarios
La modificación de archivos binarios es un poco más complicada porque generalmente tienes que leer todo el contenido, modificarlo en memoria y luego volver a escribirlo. Aquí hay un ejemplo simple que modifica un campo en una estructura:

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

struct Datos {
    int id;
    float valor;
};

int main() {
    std::fstream file("archivo.bin", std::ios::in | std::ios::out | std::ios::binary); // 
Abrir archivo binario para lectura y escritura
    if (!file) {
        std::cerr << "No se pudo abrir el archivo." << std::endl;
        return 1;
    }

    Datos d;
    file.read(reinterpret_cast<char*>(&d), sizeof(d)); // Leer datos del archivo
    d.valor = 45.6; // Modificar el valor

    file.seekp(0); // Volver al inicio del archivo
    file.write(reinterpret_cast<const char*>(&d), sizeof(d)); // Reescribir datos 
modificados
    file.close(); // Cerrar el archivo
    return 0;
}
```

Borrado de Archivos Binarios
La eliminación de archivos binarios se realiza de la misma manera que los archivos de texto:

```
#include <iostream>
#include <cstdio> // Para std::remove

int main() {
    if (std::remove("archivo.bin") != 0) { // Intentar eliminar el archivo
        std::cerr << "No se pudo eliminar el archivo." << std::endl;
    } else {
        std::cout << "Archivo eliminado exitosamente." << std::endl;
    }
    return 0;
}
```

==========
USOS:
==========
La elección entre archivos de texto y binarios en C++ depende del contexto y de los requisitos específicos de tu aplicación. Aquí te explico los casos en los que cada tipo de archivo puede ser útil:

Archivos de Texto
Ventajas:

Legibilidad Humana: Los archivos de texto son fáciles de leer y editar manualmente con editores de texto. Esto es útil para configuraciones, logs y datos que necesitan ser revisados o editados por humanos.
Portabilidad: Los archivos de texto suelen ser más portables entre diferentes sistemas operativos y plataformas, ya que el formato es más estandarizado y menos dependiente de la arquitectura del sistema.
Interoperabilidad: Muchos sistemas y lenguajes de programación manejan fácilmente los archivos de texto, facilitando el intercambio de datos entre diferentes aplicaciones y plataformas.
Simplicidad: La manipulación de archivos de texto suele ser más simple en cuanto a la escritura y lectura de datos en formato claro, como CSV o JSON.
Casos de Uso Comunes:
Configuración de Aplicaciones: Archivos .cfg, .ini, o .json para almacenar configuraciones.
Logs de Eventos: Archivos .log para registrar eventos y errores en aplicaciones.
Datos de Usuario: Archivos .txt para guardar información simple que puede ser fácilmente revisada y editada por los usuarios.
Ejemplo: Guardar una lista de usuarios y sus preferencias en un archivo de configuración en formato JSON.
Archivos Binarios
Ventajas:
Eficiencia en el Tamaño: Los archivos binarios suelen ser más compactos, ya que no necesitan almacenar delimitadores o metadatos adicionales que suelen estar presentes en archivos de texto.
Rendimiento: La lectura y escritura de archivos binarios pueden ser más rápidas porque los datos se almacenan en su formato nativo, sin necesidad de convertir entre formatos de texto y datos binarios.
Estructuración de Datos: Los archivos binarios permiten almacenar datos complejos y estructuras de datos como clases y objetos directamente, lo que puede simplificar la persistencia de datos complejos.
Integridad de Datos: Al ser menos propensos a la corrupción de datos por cambios manuales, los archivos binarios pueden ser más adecuados para aplicaciones críticas.
Casos de Uso Comunes:
Persistencia de Datos Complejos: Almacenar estados de objetos o estructuras en aplicaciones que necesitan guardar y restaurar datos complejos, como juegos o aplicaciones de simulación.
Almacenamiento de Datos de Imágenes o Multimedia: Guardar imágenes, vídeos o datos de audio en formatos binarios específicos.
Bases de Datos: Algunos sistemas de bases de datos pueden usar archivos binarios para almacenar datos y metadatos.
Ejemplo: Guardar el estado de un objeto de juego con sus propiedades complejas en un archivo binario para permitir la carga y guarda eficiente del progreso del juego.

Resumen

Archivos de Texto: Utiliza cuando necesitas legibilidad humana, portabilidad y simplicidad en la manipulación de datos. Ideal para configuraciones, logs y datos simples.
Archivos Binarios: Utiliza cuando necesitas eficiencia en el tamaño y rendimiento, y cuando trabajas con datos complejos o grandes volúmenes de datos. Ideal para aplicaciones que requieren almacenamiento y procesamiento rápido de datos binarios.
La elección entre texto y binario dependerá de las necesidades específicas de tu aplicación en términos de rendimiento, tamaño de los datos y facilidad de uso.