2001: Odisea Espacial en la Memoria

Armando Leonel Yañez Arévalo

- Tipo de actividad: Tarea.
- Materia: Programación 2.
- Fecha: 04/05/2024

En este documento se resumirá el capitulo 12 del libro **“C++, Notes for Professionals”**, el cual explica principalmente el uso de las clases *istream, ostream y streambuf,* las cuales tienen como principal función escribir y leer archivos, otro tema que aborda es el uso de los operadores “<<” y “>>”, los cuales son de entrada y salida de datos, a continuación, se revisara cada punto más a detalle:

- **Escribir en un archivo**

  Para este punto prácticamente el texto menciona que hay diferentes formas de escribir en un archivo pero que la forma mas sencilla es utilizando ***ofstream*** junto al operador **<<** el cual se encarga de **insertar la información en el archivo.**

  Ejemplo:

```
std::ofstream os("foo.txt");
if (os.is_open()) {
    os << "¡Hola Mundo!";
}
```

  Ejemplo explicado:

  // Se crea un objeto ***'os'*** de tipo ***'std::ofstream'*** que se utilizará para escribir en un archivo llamado ***"foo.txt".***

```
std::ofstream os("foo.txt");
```

  // Se verifica si el archivo se abrió correctamente.

```
if (os.is_open()) {
```

  // Si el archivo está abierto, se escribe la cadena **"¡Hola Mundo!"** en el archivo utilizando el operador de inserción **'<<'.**

```
    os << "¡Hola Mundo!";
}
```

También se nos da una alternativa si es que no queremos utilizar el operador ‘<<’ la cual es utilizando ‘write()’, a continuación un ejemplo de como utilizarlo.

Ejemplo:

```
std::ofstream os("foo.txt");
if (os.is_open()) {
    char data[] = "Foo";
    os.write(data, 3);
}
```

Ejemplo explicado:

// Se crea un objeto 'os' de tipo 'std::ofstream' que se utilizará para escribir en un archivo llamado "foo.txt".

```
std::ofstream os("foo.txt");
```

// Se verifica si el archivo se abrió correctamente.

```
if (os.is_open()) {
```

// Se declara un arreglo de caracteres llamado 'data' con la cadena "Foo".

```
 char data[] = "Foo";
```

// Se utiliza la función miembro 'write()' de 'std::ofstream' para escribir 3 caracteres desde 'data' en el archivo.

```
os.write(data, 3); // Esto escribe los primeros 3 caracteres de 'data' en el archivo "foo.txt".
}
```

En resumen, este código abre un archivo llamado "foo.txt", verifica si se abrió correctamente y, si es así, escribe los primeros 3 caracteres de la cadena "Foo" en el archivo.

Por último, para esta parte, también se menciona sobre la importancia de verificar en caso de errores cada vez que se escriba en un archivo, esto es muy fácil de hacer ya que prácticamente es solo llamar la función.

Ejemplo:

```
os << "¡Hola Badbit!";
if (os.bad()) // Si llega a fallar se activa…
```

Prácticamente es todo lo que se habla en la parte de escribir en un archivo, aunque hay otras cosas aparte que hay que tomar en cuenta para aprovechar al máximo esta clase.

- **Abrir un archivo**

  Es importante saber que los archivos de abren de la misma forma para los 3 tipos de clase, y se utiliza el siguiente constructor:

  Ejemplo:

```
// ifstream: Abre el archivo "foo.txt" para lectura.
std::ifstream ifs("foo.txt");
// ofstream: Abre el archivo "foo.txt" para escritura.
std::ofstream ofs("foo.txt");
// fstream: Abre el archivo "foo.txt" para lectura y escritura.
std::fstream iofs("foo.txt");
```

  Aunque también se puede abrir de la siguiente forma:

```
std::ifstream ifs; //lectura
ifs.open("bar.txt");

std::ofstream ofs; //escritura
ofs.open("bar.txt");

std::fstream iofs; //lectura y lectura
iofs.open("bar.txt");
```

  Es importante tener en cuenta que siempre se debe verificar que el archivo pueda abrirse, ya que hay muchos fallos que pueden presentarse en el proceso, por tal

motivo es importante incluir la verificación, la cual se hace de la siguiente manera:

```
std::ifstream ifs("fooo.txt");
//En esta parte se hace la verificación y en caso de no abrirse se imprime el mensaje.
if (!ifs.is_open()) {
    throw CustomException(ifs, "No se pudo abrir el archivo");
}
```

Por último, en esta parte del texto se menciona como abrir archivos en caso de contener barras invertidas, nos da los siguientes casos:

```
// Abre el archivo 'c:\\folder\\foo.txt' en Windows.
std::ifstream ifs("c:\\\\folder\\\\foo.txt");

// Abre el archivo 'c:\\folder\\foo.txt' en Windows.
std::ifstream ifs(R"(c:\\folder\\foo.txt)");

// Abre el archivo 'c:\\folder\\foo.txt' en Windows.
std::ifstream ifs("c:/folder/foo.txt");
```

- **Leer un archivo**

  Existen muchas formas de leer un archivo, una de las mas practicas es utilizando el operador “>>” para extraer datos. Uno de los ejemplos para la extracción es el siguiente.

  Ejemplo:

  Suponiendo que en el archivo se tienen los datos
  “John Doe 25 4 6 1987”

  Se puede utilizar este código para leer los datos:

```
std::ifstream is("foo.txt");
std::string firstname, lastname;
int age, bmonth, bday, byear;
while (is >> firstname >> lastname >> age >> bmonth >> bday >> byear)
```

Prácticamente cómo funciona este código es que el operador de extracción de flujo >> extrae cada carácter y se detiene si encuentra un carácter que no se puede almacenar o si es un carácter especial, estos pueden ser:

- Para tipos de cadena, el operador se detiene en un espacio en blanco () o en una nueva línea (\n).

- Para números, el operador se detiene en un carácter que no es un número.

Si se desea leer un archivo completo como cadena se puede usar el código:

```
// Abre 'foo.txt'.
std::ifstream is("foo.txt");
std::string whole_file;
// Establece la posición al final del archivo.
is.seekg(0, std::ios::end);
// Reserva memoria para el archivo.
whole_file.reserve(is.tellg());
// Establece la posición al principio del archivo.
is.seekg(0, std::ios::beg);
// Establece el contenido de 'whole_file' a todos los caracteres en el archivo.
whole_file.assign(std::istreambuf_iterator<char>(is),
                  std::istreambuf_iterator<char>());
```

También si se desea leer un archivo línea a línea se puede hacer de la siguiente forma:

```
std::ifstream is("foo.txt");
// La función getline devuelve false si no hay más líneas.
```

```
for (std::string str; std::getline(is, str);) {
// Procesa la línea que se ha leído.
}
```

También se puede leer una cantidad fija de caracteres, eso se puede hacer de la siguiente forma:

```
std::ifstream is("foo.txt");
char str[4];
// Lee 4 caracteres del archivo.
is.read(str, 4)
```

Como se mencionó anteriormente, es importante verificar que el comando se ejecuto correctamente por este motivo también es importante leer si la lectura fue exitosa, esto se puede hacer con el siguiente código:

```
is.read(str, 4);
if (is.fail()) // En caso de fallar se ejecuta.
```

- **Modos de apertura**

  Un modo de apertura es un ajuste para controlar como el flujo puede abrir un archivo, es importante mencionar que todos los modos se pueden encontrar en el espacio de nombres ***std::ios***.

  El modo de apertura se puede especificar como segundo parámetro del constructor o de la función ***open(),*** ejemplo:

```
std::ofstream os("foo.txt", std::ios::out | std::ios::trunc);
std::ifstream is;
is.open("foo.txt", std::ios::in | std::ios::binary);
```

Otra cosa que se debe tener en cuenta es establecer ***ios::in*** o ***ios::out*** si deseas establecer otros indicadores, de caso contrario se establecerán valores por defecto.

Esta tabla muestra los modos de apertura que se le puede especificar:

Modo	Significado	Para	Descripción
app	añadir	Salida	Agrega datos al final del archivo.
binary	binario	Entrada/Salida	La entrada y salida se realizan en binario.
in	entrada	Entrada	Abre el archivo para lectura.
out	salida	Salida	Abre el archivo para escritura.
trunc	truncar	Entrada/Salida	Elimina el contenido del archivo al abrirlo.
ate	al final	Entrada	Va al final del archivo al abrirlo.

- **Leer un archivo ASCII**

  Existen 3 formas distintas de leer un archivo ***ASCII*** en un objeto ***string***, estos son los tipos que se mencionan:

  1- Utilizando un 'std::stringstream':

```
std::ifstream f("file.txt");
if (f) {
    std::stringstream buffer;
    buffer << f.rdbuf();
    f.close();
}
```

  2- Utilizando un constructor de ***'std::string'*** con iteradores:

```
std::ifstream f("file.txt");
if (f) {
    std::string str((std::istreambuf_iterator<char>(f)),
```

```
            std::istreambuf_iterator<char>());
}
```

3- Utilizando manipulaciones directas de flujo para calcular el tamaño del archivo y luego leerlo completo:

```
std::ifstream f("file.txt");
if (f) {
    f.seekg(0, std::ios::end);
    const auto size = f.tellg();
    std::string str(size, ' ');
    f.seekg(0);
    f.read(&str[0], size);
    f.close();
}
```

- **Escribir archivos con configuraciones de localización no estándar**

  Si se llega a necesitar escribir un archivo con configuraciones de localización diferentes se puede usar 'std::locale y std::basic_ios::imbue()', este es el código de ejemplo que se nos da:

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

int main() {
std::cout << "La configuración de localización preferida del usuario es "
          << std::locale("").name().c_str() << std::endl;

// Escribe un valor de punto flotante utilizando la localización preferida del usuario.
std::ofstream ofs1;
ofs1.imbue(std::locale(""));
ofs1.open("file1.txt");
ofs1 << 78123.456 << std::endl;
```

```
// Utiliza una localización específica (los nombres dependen del sistema)
std::ofstream ofs2;
ofs2.imbue(std::locale("en_US.UTF-8"));
ofs2.open("file2.txt");
ofs2 << 78123.456 << std::endl;

// Cambia a la localización clásica "C"
std::ofstream ofs3;
ofs3.imbue(std::locale::classic());
ofs3.open("file3.txt");
ofs3 << 78123.456 << std::endl;
}
```

Este tipo de prácticas son útiles si tu programa utiliza una localización predeterminada diferente y deseas asegurar un estándar fijo para leer y escribir archivos.

- **¿Es una mala práctica verificar el final del archivo dentro de una condición de bucle?**

Esta parte es simple, solo explica el funcionamiento de ***'eof'*** el cual devuelve true solo después de leer el final del archivo, este es un ejemplo de su utilización.

```
while (!f.eof())
{
    f >> buffer >> std::ws;
    if (f.fail())
        break;
    /* Usa `buffer` */
}
```

Hay otras formas de hacerlo, como puede ser también:

```
while (f >> buffer)
{
    /* Usa `buffer` */
}
```

Esta es la forma más simple y más recomendada que hay.

- **Vaciar un flujo**

  Los flujos de archivo están en búfer por defecto, lo que significa que las escrituras en el flujo pueden no reflejarse de inmediato en el archivo, por esta razón es importante saber como forzar las escrituras, esto se puede realizar de las siguientes dos formas:

```
os << "¡Hola Mundo!\n" << std::flush;
os << "¡Hola Mundo!" << std::endl;
```

  Ejemplo:

```
std::ofstream os("foo.txt");
os << "¡Hola Mundo!" << std::flush;
char data[3] = "Foo";
os.write(data, 3);
os.flush();
```

- **Lectura de un archivo en un contenedor**

  Existen distintas formas de hacer esto, una es utilizando "***std::string y >>"***, también hay otra opción usando ***"std::istream_iterator"*** y por ultimo explica como usar ***getline*** para obtener líneas completas en lugar de palabras.

  Ejemplo *std::string y* >>:

```
std::ifstream file("file3.txt");
std::vector<std::string> v;
std::string s;
while (file >> s)
{
    v.push_back(s);
```

```
}
```

Ejemplo *std::istream_iterator*:

```
std::ifstream file("file3.txt");
std::vector<std::string> v(std::istream_iterator<std::string>{file},
                           std::istream_iterator<std::string>{});
```

Ejemplo *getline*:

```
struct Line
{
    // Almacenar datos aquí
    std::string data;
    // Convertir objeto a cadena
    operator std::string const&() const { return data; }
    // Leer una línea de un flujo.
    friend std::istream& operator>>(std::istream& stream, Line& line)
    {
        return std::getline(stream, line.data);
    }
};

std::ifstream file("file3.txt");
// Leer las líneas de un archivo en un contenedor.
std::vector<std::string> v(std::istream_iterator<Line>{file},
                           std::istream_iterator<Line>{});
```

- **Copiar un archivo**

  Como el subtitulo lo indica, simplemente es saber como copiar un archivo, esto se puede hacer de la siguiente forma:

```
#include <filesystem>
std::filesystem::copy_file("source_filename", "dest_filename");
//Se colocará archivo de donde se copiará y el nuevo archivo donde se copiará.
```

- **Cerrar un archivo**

  Aunque cerrar un archivo de forma explicita es raramente necesario, ya que en teoría se cierran automáticamente, pero es importante limitar el tiempo de vida de un objeto para no mantenerlo abierto demasiado tiempo, esto se puede hacer usando **{()}** como se muestra en el ejemplo:

```
std::string const datos_preparados = preparar_datos();
{
    // Abrir un archivo para escribir.
    std::ofstream salida("foo.txt");
    // Escribir datos.
    salida << datos_preparados;
} // El ofstream saldrá de alcance aquí.
```

  Aunque esta es la opción mas recomendable, también hay la opción de cerrar utilizando ***close();*** por ejemplo:

```
// Abrir el archivo "foo.txt" por primera vez.
std::ofstream salida("foo.txt");
// Obtener algunos datos para escribir de alguna parte.
std::string const datos_preparados = preparar_datos();
// Escribir datos en el archivo "foo.txt".
salida << datos_preparados;
// Cerrar el archivo "foo.txt".
salida.close();
```

- **Lectura de una estructura desde un archivo de texto formateado**

  Este es un ejemplo de como leer una estructura ***(info_type)*** desde un archivo de texto formateado en C++, esto se logra utilizando el operador >> para permitir la lectura formateada de una instancia de ***info_type*** desde un flujo de entrada. Luego, se muestra un ejemplo de cómo utilizar esta estructura para leer datos desde un archivo de texto (file4.txt) para posteriormente imprimirla, ejemplo:

```
struct info_type
```

```
{
    std::string name;
    int age;
    float height;

    friend std::istream& operator>>(std::istream& is, info_type& info)
    {
        // saltar espacios en blanco
        is >> std::ws;
        std::getline(is, info.name);
        is >> info.age;
        is >> info.height;
        return is;
    }
};

void func4()
{
    auto file = std::ifstream("file4.txt");
    std::vector<info_type> v;
    for (info_type info; file >> info;) // seguir leyendo hasta que se acabe
    {
        // solo llegamos aquí si la lectura tuvo éxito
        v.push_back(info);
    }
    for (auto const& info : v)
    {
        std::cout << " name: " << info.name << '\n';
        std::cout << " age: " << info.age << " años" << '\n';
        std::cout << "height: " << info.height << "lbs" << '\n';
        std::cout << '\n';
    }
}
```

Prácticamente esto es el mayor resumen que se puede hacer del documento, ya que realmente lo mas importante del documento son las sintaxis y ejemplos que se nos dan para lograr hacer las operaciones de lectura y escritura de datos. El libro es muy bueno, da ejemplos muy entendibles, pero debo mencionar que también deja muy débil la parte de la teoría ya que no explica a profundidad muchas de las cosas que se están utilizando, igual se puede entender ya que esto es mas practica que nada, pero si tiene ese punto débil que al menos yo identifique.

En conclusión es un tema muy interesante y creo que de lo mas funcional ya que esto nos puede ayudar de muchas formas, desde crear pequeños sistemas de encriptación, hasta crear bases de datos locales con archivos de texto, se me hizo muy interesante que literal puedes entrar a cualquier archivo en cualquier locación mientras el comando sea el adecuado, esto creo que es una funcionalidad muy poderosa, en si el tema es interesante, tengo varias dudas sobre el funcionamiento de algunas cosas, pero son dudas que podre resolver hasta probar estas clases.

- **Fuente:**

[1] C++ notes for professionals. goalkicker.com, 2022. [PDF]. Disponible en: https://goalkicker.com/CPlusPlusBook/