

File Streams

Now, we shall learn how to communicate with files using C++.

File streams enable us to work with files. They need the header `<fstream>`. The file streams automatically manage their file for its whole lifetime.

Whether we use a file stream for input, output, or with the character type `char` or `wchar_t` there are various file stream classes:

Class	Use
<code>std::ifstream</code> and <code>std::wifstream</code>	File stream for the input of data of type <code>char</code> and <code>wchar_t</code> .
<code>std::ofstream</code> and <code>std::wofstream</code>	File stream for the output of data of type <code>char</code> and <code>wchar_t</code> .
<code>std::fstream</code> and <code>std::wfstream</code>	File stream for the input and output of data of type <code>char</code> and <code>wchar_t</code> .
<code>std::filebuf</code> and <code>std::wfilebuf</code>	Data buffer of type <code>char</code> and <code>wchar_t</code> .

Set the file position pointer

File streams used for reading and writing have to set the file position pointer after the contents change.

Flags enable us to set the opening mode of a file stream.

Flag	Description
<code>std::ios::in</code>	Opens the file stream for reading (default for <code>std::ifstream</code> and <code>std::wifstream</code>).
<code>std::ios::out</code>	Opens the file stream for writing (default for <code>std::ofstream</code> and <code>std::wofstream</code>).
<code>std::ios::app</code>	Appends the character to the end of the file stream.
<code>std::ios::ate</code>	Sets the initial position of the file position pointer at the end of the file stream.
<code>std::ios::trunc</code>	Deletes the original file.
<code>std::ios::binary</code>	Suppresses the interpretation of an escape sequence in the file stream.

Flags for the opening of a file stream

It's quite easy to copy the file named `in` to the file named `out` with the file buffer `in.rdbuf()`. The error handling is missing in this short example.

```
#include <fstream>
...
std::ifstream in("inFile.txt");
std::ofstream out("outFile.txt");
out << in.rdbuf();
```



If we combine the C++ flags, we can compare the C++ and C modes to open a file.

C++ mode	Description	C mode
----------	-------------	--------

<code>std::ios::in</code>	Reads the file.	<code>"r"</code>
<code>std::ios::out</code>	Writes the file.	<code>"w"</code>
<code>std::ios::out std::ios::app</code>	Appends to the file.	<code>"a"</code>
<code>std::ios::in std::ios::out</code>	Reads and writes the file.	<code>"r+"</code>
<code>std::ios::in std::ios::out std::ios::trunc</code>	Writes and reads the file.	<code>"w+"</code>

Opening of a file with C++ and C

The file has to exist with the mode `"r"` and `"r+"`. On the contrary, the file is being created with `"a"` and `"w+"`. The file is overwritten with `"w"`.

We can explicitly manage the lifetime of a file stream.

Flag	Description
<code>infile.open(name)</code>	Opens the file <code>name</code> for reading.
<code>infile.open(name, flags)</code>	Opens the file <code>name</code> with the flags <code>flags</code> for reading.
<code>infile.close()</code>	Closes the file.
<code>infile.is_open()</code>	Checks if the file is open.

Managing the lifetime of a file stream

Random access enables us to set the file position's pointer arbitrarily.

When a file stream is constructed, the file's position pointer points to the

When a file stream is constructed, the file position pointer points to the beginning of the file. We can adjust the position with the methods of the file stream `file`.

Method	Description
<code>file.tellg()</code>	Returns the read position of <code>file</code> .
<code>file.tellp()</code>	Returns the write position of <code>file</code> .
<code>file.seekg(pos)</code>	Sets the read position of <code>file</code> to <code>pos</code> .
<code>file.seekp(pos)</code>	Sets the write position of <code>file</code> to <code>pos</code> .
<code>file.seekg(off, rpos)</code>	Sets the read position of <code>file</code> to the offset <code>off</code> relative to <code>rpos</code> .
<code>file.seekp(off, rpos)</code>	Sets the write position of <code>file</code> to the offset <code>off</code> relative to <code>rpos</code> .

Navigate in a file stream

`off` has to be a number. `rpos` can have three values:

<code>rpos</code> value	Description
<code>std::ios::beg</code>	Position at the beginning of the file.
<code>std::ios::cur</code>	Position at the current position.
<code>std::ios::end</code>	Position at the end of the file.

⚠️ Respect the file boundaries

If we randomly access a file, the C++ runtime does not check the file boundaries. Reading or writing data outside the boundaries is an *undefined behavior*.

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

int writeFile(const std::string name){

    std::ofstream outFile(name);

    if (!outFile){
        std::cerr << "Could not open file " << name << std::endl;
        exit(1);
    }

    for ( unsigned int i=0; i < 10; ++i){
        outFile << i << "          0123456789" << std::endl;
    }
}

int main(){

    std::cout << std::endl;

    std::string random{"random.txt"};

    writeFile(random);

    std::ifstream inFile(random);

    if (!inFile){
        std::cerr << "Could not open file " << random << std::endl;
        exit(1);
    }

    std::string line;

    std::cout << "The whole file : " << std::endl;
    std::cout << inFile.rdbuf();
    std::cout << "inFile.tellg(): " << inFile.tellg() << std::endl;

    std::cout << std::endl;

    inFile.seekg(0);
    inFile.seekg(0, std::ios::beg); // redundant
    getline(inFile, line);
    std::cout << line << std::endl;

    inFile.seekg(20, std::ios::cur);
    getline(inFile, line);
    std::cout << line << std::endl;

    inFile.seekg( 20, std::ios::end);
```

```
inFile.seekg(-20, std::ios::end),  
getline(inFile, line);  
std::cout << line << std::endl;  
  
std::cout << std::endl;  
}
```



Random access

We'll solve an exercise to get a better grip on this concept.