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

△ 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
std::ios::in	Opens the file stream for reading (default for std::ifstream and std::wifstream).
std::ios::out	Opens the file stream for writing (default for std::ofstream and std::wofstream).
std::ios::app	Appends the character to the end of the file stream.
std::ios::ate	Sets the initial position of the file position pointer at the end of the file stream.
std::ios::trunc	Deletes the original file.
std::ios::binary	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();</pre>
```

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

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

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
<pre>infile.open(name)</pre>	Opens the file name for reading.
<pre>infile.open(name, flags)</pre>	Opens the file name with the flags for reading.
<pre>infile.close()</pre>	Closes the file.
<pre>infile.is_open()</pre>	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

beginning of the file. We can adjust the position with the methods of the file stream file.

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

Navigate in a file stream

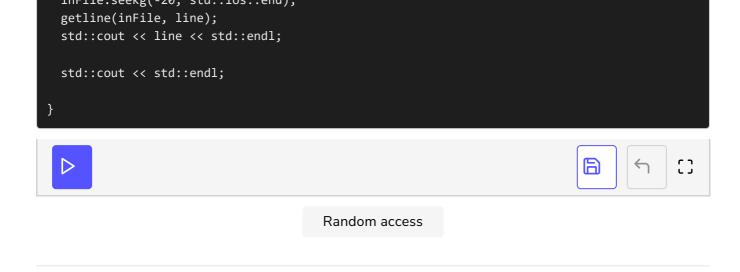
off has to be a number. rpos can have three values:

rpos value	Description
std::ios::beg	Position at the beginning of the file.
std::ios::cur	Position at the current position.
std::ios::end	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;</pre>
    exit(1);
  for ( unsigned int i=0; i < 10; ++i){
    outFile << i << " 0123456789" << std::endl;
}
int main(){
  std::cout << std::endl;</pre>
  std::string random{"random.txt"};
  writeFile(random);
  std::ifstream inFile(random);
  if (!inFile){
   std::cerr << "Could not open file " << random << std::endl;</pre>
    exit(1);
  }
  std::string line;
  std::cout << "The whole file : " << std::endl;</pre>
  std::cout << inFile.rdbuf();</pre>
  std::cout << "inFile.tellg(): " << inFile.tellg() << std::endl;</pre>
  std::cout << std::endl;</pre>
  inFile.seekg(0);
  inFile.seekg(0, std::ios::beg); // redundant
  getline(inFile, line);
  std::cout << line << std::endl;</pre>
  inFile.seekg(20, std::ios::cur);
  getline(inFile, line);
  std::cout << line << std::endl;</pre>
  infile scales/ 20
```



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