

## 28.6 — Basic file I/O

👤 [ALEX<sup>1</sup>](#) ⌚ FEBRUARY 27, 2024

File I/O in C++ works very similarly to normal I/O (with a few minor added complexities). There are 3 basic file I/O classes in C++: `ifstream` (derived from `istream`), `ofstream` (derived from `ostream`), and `fstream` (derived from `iostream`). These classes do file input, output, and input/output respectively. To use the file I/O classes, you will need to include the `fstream` header.

Unlike the `cout`, `cin`, `cerr`, and `clog` streams, which are already ready for use, file streams have to be explicitly set up by the programmer. However, this is extremely simple: to open a file for reading and/or writing, simply instantiate an object of the appropriate file I/O class, with the name of the file as a parameter. Then use the insertion (`<<`) or extraction (`>>`) operator to write to or read data from the file. Once you are done, there are several ways to close a file: explicitly call the `close()` function, or just let the file I/O variable go out of scope (the file I/O class destructor will close the file for you).

### File output

To do file output in the following example, we're going to use the `ofstream` class. This is extremely straightforward:

```
1  #include <fstream>
2  #include <iostream>
3
4  int main()
5  {
6      // ofstream is used for writing files
7      // We'll make a file called Sample.txt
8      std::ofstream outf{ "Sample.txt" };
9
10     // If we couldn't open the output file stream for writing
11     if (!outf)
12     {
13         // Print an error and exit
14         std::cerr << "Uh oh, Sample.txt could not be opened for writing!\n";
15         return 1;
16     }
17
18     // We'll write two lines into this file
19     outf << "This is line 1\n";
20     outf << "This is line 2\n";
21
22     return 0;
23
24     // When outf goes out of scope, the ofstream
25     // destructor will close the file
26 }
```

If you look in your project directory, you should see a file called `Sample.txt`. If you open it with a text editor, you will see that it indeed contains two lines we wrote to the file.

Note that it is also possible to use the `put()` function to write a single character to the file.

## File input

Now, we'll take the file we wrote in the last example and read it back in from disk. Note that `ifstream` returns a 0 if we've reached the end of the file (EOF). We'll use this fact to determine how much to read.

```
1  #include <fstream>
2  #include <iostream>
3  #include <string>
4
5  int main()
6  {
7      // ifstream is used for reading files
8      // We'll read from a file called Sample.txt
9      std::ifstream inf{ "Sample.txt" };
10
11     // If we couldn't open the output file stream for reading
12     if (!inf)
13     {
14         // Print an error and exit
15         std::cerr << "Uh oh, Sample.txt could not be opened for reading!\n";
16         return 1;
17     }
18
19     // While there's still stuff left to read
20     std::string strInput{};
21     while (inf >> strInput)
22         std::cout << strInput << '\n';
23
24     return 0;
25
26     // When inf goes out of scope, the ifstream
27     // destructor will close the file
28 }
```

This produces the result:

```
This
is
line
1
This
is
line
2
```

Hmmm, that wasn't quite what we wanted. Remember that the extraction operator breaks on whitespace. In order to read in entire lines, we'll have to use the `getline()` function.

```

1  #include <fstream>
2  #include <iostream>
3  #include <string>
4
5  int main()
6  {
7      // ifstream is used for reading files
8      // We'll read from a file called Sample.txt
9      std::ifstream inf{ "Sample.txt" };
10
11     // If we couldn't open the input file stream for reading
12     if (!inf)
13     {
14         // Print an error and exit
15         std::cerr << "Uh oh, Sample.txt could not be opened for reading!\n";
16         return 1;
17     }
18
19     // While there's still stuff left to read
20     std::string strInput{};
21     while (std::getline(inf, strInput))
22         std::cout << strInput << '\n';
23
24     return 0;
25
26     // When inf goes out of scope, the ifstream
27     // destructor will close the file
28 }

```

This produces the result:

```

This is line 1
This is line 2

```

## Buffered output

Output in C++ may be buffered. This means that anything that is output to a file stream may not be written to disk immediately. Instead, several output operations may be batched and handled together. This is done primarily for performance reasons. When a buffer is written to disk, this is called **flushing** the buffer. One way to cause the buffer to be flushed is to close the file -- the contents of the buffer will be flushed to disk, and then the file will be closed.

Buffering is usually not a problem, but in certain circumstance it can cause complications for the unwary. The main culprit in this case is when there is data in the buffer, and then program terminates immediately (either by crashing, or by calling `exit()`). In these cases, the destructors for the file stream classes are not executed, which means the files are never closed, which means the buffers are never flushed. In this case, the data in the buffer is not written to disk, and is lost forever. This is why it is always a good idea to explicitly close any open files before calling `exit()`.

It is possible to flush the buffer manually using the `ostream::flush()` function or sending `std::flush` to the output stream. Either of these methods can be useful to ensure the contents of the buffer are written to disk immediately, just in case the program crashes.

One interesting note is that `std::endl` also flushes the output stream. Consequently, overuse of `std::endl` (causing unnecessary buffer flushes) can have performance impacts when doing buffered I/O where flushes are expensive (such as writing to a file). For this reason, performance conscious programmers will often use `\n` instead of `std::endl` to insert a newline into the output stream, to avoid unnecessary flushing of the buffer.

## File modes

What happens if we try to write to a file that already exists? Running the output example again shows that the original file is completely overwritten each time the program is run. What if, instead, we wanted to append some more data to the end of the file? It turns out that the file stream constructors take an optional second parameter that allows you to specify information about how the file should be opened. This parameter is called mode, and the valid flags that it accepts live in the `ios` class.

ios file mode	Meaning
app	Opens the file in append mode
ate	Seeks to the end of the file before reading/writing
binary	Opens the file in binary mode (instead of text mode)
in	Opens the file in read mode (default for ifstream)
out	Opens the file in write mode (default for ofstream)
trunc	Erases the file if it already exists

It is possible to specify multiple flags by bitwise ORing them together (using the `|` operator). `ifstream` defaults to `std::ios::in` file mode. `ofstream` defaults to `std::ios::out` file mode. And `fstream` defaults to `std::ios::in | std::ios::out` file mode, meaning you can both read and write by default.

### Tip

Due to the way `fstream` was designed, it may fail if `std::ios::in` is used and the file being opened does not exist. If you need to create a new file using `fstream`, use `std::ios::out` mode only.

Let's write a program that appends two more lines to the `Sample.txt` file we previously created:

```
1  #include <iostream>
2  #include <fstream>
3
4  int main()
5  {
6      // We'll pass the ios::app flag to tell the ofstream to append
7      // rather than rewrite the file. We do not need to pass in std::ios::out
8      // because ofstream defaults to std::ios::out
9      std::ofstream outf{ "Sample.txt", std::ios::app };
10
11     // If we couldn't open the output file stream for writing
12     if (!outf)
13     {
14         // Print an error and exit
15         std::cerr << "Uh oh, Sample.txt could not be opened for writing!\n";
16         return 1;
17     }
18
19     outf << "This is line 3\n";
20     outf << "This is line 4\n";
21
22     return 0;
23
24     // When outf goes out of scope, the ofstream
25     // destructor will close the file
26 }
```

Now if we take a look at Sample.txt (using one of the above sample programs that prints its contents, or loading it in a text editor), we will see the following:

```
This is line 1
This is line 2
This is line 3
This is line 4
```

### Explicitly opening files using open()

Just like it is possible to explicitly close a file stream using close(), it's also possible to explicitly open a file stream using open(). open() works just like the file stream constructors -- it takes a file name and an optional file mode.

For example:

```
1 std::ofstream outf{ "Sample.txt" };
2 outf << "This is line 1\n";
3 outf << "This is line 2\n";
4 outf.close(); // explicitly close the file
5
6 // Oops, we forgot something
7 outf.open("Sample.txt", std::ios::app);
8 outf << "This is line 3\n";
9 outf.close();
```

You can find more information about the open() function [here](https://en.cppreference.com/w/cpp/io/basic_filebuf/open) ([https://en.cppreference.com/w/cpp/io/basic\\_filebuf/open](https://en.cppreference.com/w/cpp/io/basic_filebuf/open))<sup>2</sup>.



**[Next lesson](#)**

**28.7** [Random file I/O](#)



**[Back to table of contents](#)**



**[Previous lesson](#)**

**28.5** [Stream states and input validation](#)



B

U

URL

INLINE CODE

C++ CODE BLOCK

HELP!

Leave a comment...

 Name\*


 Email\* 

Notify me about replies:



POST COMMENT

 Find a mistake? Leave a comment above! 

 Avatars from <https://gravatar.com/><sup>8</sup> are connected to your provided email address.

198 COMMENTS

Newest ▼



**Nidhi Gupta**

 February 23, 2025 9:59 pm PST

File I/O in C++ is similar to regular I/O, only with additional steps and complexities. There are three primary classes used for file operations: ifstream for reading, ofstream for writing, and fstream for reading and writing, all of which require the fstream header to be included. Unlike the default streams (cin, cout, etc.), file streams must be opened by the programmer explicitly and closed either by calling close() or by letting the stream go out of scope. Data is output using the insertion operator or procedures like put(), and input with the extraction operator or getline() to read complete lines. Output is usually buffered to maximize performance, so data is not necessarily written to disk immediately unless the buffer is manually flushed or on closing the stream. File streams also support a range of modes—append, binary, and truncation—that can be combined using bitwise OR to tailor file handling behavior.

 1  Reply



**Nidhi Gupta**

 February 23, 2025 9:58 pm PST

this is cool

 0  Reply



**Leni**

 February 23, 2025 6:11 pm PST

File I/O in C++ provides flexible ways to read and write data, but understanding file modes is crucial to avoid unintended overwrites. How do different file modes impact file operations?

 2  Reply



Faizan

🕒 June 28, 2024 4:28 am PDT

And now I can actually do Advent of Code in C++



4



Reply



D D

🕒 February 26, 2024 8:40 am PST

Hello, Alex

1. We can change this in the first example:

```
1 while (inf)
2 {
3
4     std::string strInput; // {} initialization is forgotten
5     inf >> strInput;
6     std::cout << strInput << '\n';
7 }
```

to this

```
1 std::string strInput{};
2 while (inf >> strInput)
3     std::cout << strInput << '\n';
```

2. The second one

```
1 while (inf)
2 {
3     std::string strInput;
4     std::getline(inf, strInput);
5     std::cout << strInput << '\n';
6 }
```

to this

```
1 std::string strInput{};
2 while (std::getline(inf, strInput))
3     std::cout << strInput << '\n';
```

📝 Last edited 1 year ago by D D



1



Reply



Alex

Author

🗨 Reply to [D D](#)<sup>9</sup> 🕒 February 27, 2024 8:19 pm PST

Thanks for this and all of the other great feedback you've left. I appreciate you.

👍 4    ➡ Reply

D D

 D D    ➡ Reply to [Alex](#)<sup>10</sup>    ⌚ March 2, 2024 11:23 pm PST

Hello, Alex. Tell me, please, when are you going to add new lessons in C++11, C++17?

👍 1    ➡ Reply



**Alex**    Author

➡ Reply to [D D](#)<sup>11</sup>    ⌚ March 4, 2024 6:47 pm PST

That question is a bit vague. I pick which lessons I write next based on where I think the biggest gaps or needs are.

👍 2    ➡ Reply



**\_eklektos\_**

⌚ December 13, 2023 11:40 pm PST

```
1 // ifstream is used for reading files
2 // We'll read from a file called Sample.txt
3 std::ifstream inf{ "Sample.txt" };
4
5 // If we couldn't open the output file stream for reading -
6 // *****I wonder if this comment is right*****
7 if (!inf)
8 {
9     // Print an error and exit
10    std::cerr << "Uh oh, Sample.txt could not be opened for reading!\n";
11    return 1;
12 }
```

✎ Last edited 1 year ago by [\\_eklektos\\_](#)

👍 0    ➡ Reply



**jhdthdkfgjkhdfg**

⌚ October 26, 2023 3:24 am PDT

In the following code:



```

1 #include <fstream>
2 #include <iostream>
3 #include <string>
4
5 const std::ifstream &getFile();
6 void printFileToConsole(const std::ifstream &inf);
7
8 int main() { printFileToConsole(getFile()); }
9
10 const std::ifstream &getFile() {
11     static std::ifstream inf{"input.txt"};
12
13     if (!inf) {
14         std::cerr << "Unable to open file.";
15     }
16
17     return inf;
18 }
19
20 void printFileToConsole(const std::ifstream &inf) {
21     while (inf) {
22         std::string strInput;
23         inf >> strInput;
24         std::cout << strInput << '\n';
25     }
26 }

```

My LSP is indicating an error line 22: invalid operands to binary expression ('const std::ifstream' and 'std::string').

But I'm only doing a reading operation on that file, so I don't really understand the error.

Thanks!

👍 0    ➡ Reply



[\\_eklektos\\_](#)

🗨 Reply to [jhdthdkfgjkhdfg](#)<sup>12</sup> 🕒 December 14, 2023 2:51 am PST

Remove the const from everywhere.

**You cannot extract from a const std::ifstream object using the >> operator because the >> operator modifies the state of the stream by extracting data from it. The const qualifier indicates that the member functions of the object cannot modify the object's(it's std::ifstream) data members.**

📝 Last edited 1 year ago by [\\_eklektos\\_](#)

👍 1    ➡ Reply



...

🕒 October 15, 2023 1:47 am PDT

Hi Alex, is there was any plan to update these 2 lessons about files to c++20 or 23 ?

📝 Last edited 1 year ago by ...

👍 0    ➡ Reply



**Alex** Author

🗨 Reply to ...<sup>13</sup>    🕒 October 17, 2023 7:40 pm PDT

Yes, but I doubt soon. Too many higher priority things ahead of this.

👍 1    ➡ Reply



**yurr**

🗨 Reply to Alex<sup>14</sup>    🕒 January 23, 2025 6:43 pm PST

```
@@BLOCK4@@#include <fstream>
#include <iostream>

int main()
{
    // ofstream is used for writing files
    // We'll make a file called Sample.txt
    std::ofstream outf{ "Sample.txt" };

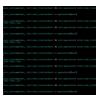
    // If we couldn't open the output file stream for writing
    if (!outf)
    {
        // Print an error and exit
        std::cerr << "Uh oh, Sample.txt could not be opened for writing!\n";
        return 1;
    }

    // We'll write two lines into this file
    outf << "This is line 1\n";
    outf << "This is line 2\n";

    return 0;

    // When outf goes out of scope, the ofstream
    // destructor will close the file
}
```

👍 0    ➡ Reply



**learnccp lesson reviewer**

🕒 August 3, 2023 11:10 am PDT

W

👍 0    ➡ Reply

Emeka Daniel

🕒 May 28, 2023 3:49 pm PDT

This lesson and the next were written pre-C++17. In C++17, file I/O is better done using `std::filesystem`.

Thanks for the info. I went exploring with the new feature and it is beyond great.

At first i thought it was a separate class that handle file I/O just like `fstream`, but then i realized it is just a helper class. I think you should rephrase the wording a bit to eliminate ambiguity :).

```
1  #include "gfunctions.h"
2  #include "Random.h"
3  #include "customTypes.h"
4  #include "miscellany.h"
5  #include "squareGame.h"
6  #include "fPuzzleGame.h"
7  #include "guessGame.h"
8  #include "logicGame.h"
9
10 // To run include <filesystem> and obviously change the folder name and root directory
11 to urs.
12
13 int main( [[maybe_unused]]int argc, [[maybe_unused]]char* argv[] )
14 {
15     namespace fs = std::filesystem;
16
17     fs::path p{"C:/Users/MIKE EMEKA/Documents/"};
18     p /= "A++/";
19     fs::create_directory(p);
20     p /= "file.txt";
21
22     if (p.empty())
23         std::cout << "object: p of type path is empty\n";
24     if(fs::exists(p))
25         std::cout << "File: " << p.filename() << " exists\n\n";
26
27     std::ofstream file;
28     file.open(p, std::ios::app | std::ios::out);
29
30     if (!file.is_open())
31     {
32         std::cerr << "Could not open file: " << p.filename() << ". \nCheck directory:
33 " << p.parent_path() << ", if file exists\n";
34     }
35
36     file << "I AM HERE! JACK\n";
37
38     std::cout << "Size of file: " << fs::file_size(p) << '\n';
39
40
41
42
43     return EXIT_SUCCESS;
44 }
```

Awesome Lesson.

👍 1    ➡ Reply



Hritik

Reply to [Emeka Daniel](#)<sup>15</sup> ⌚ October 14, 2023 12:01 am PDT

Hey There , is there any detailed tutorials which cover <filesystem> ?

👍 0

➡ Reply



Emeka Daniel

Reply to [Hritik](#)<sup>16</sup> ⌚ October 14, 2023 12:31 am PDT

As of the time of writing this comment, I did not know of one, I just got the information of how to use certain classes and functions from [cppreference.com](https://en.cppreference.com/w/cpp/filesystem) (<https://en.cppreference.com/w/cpp/filesystem>)<sup>17</sup>, but recently I learned that the C++17 library add on was gotten from the boost library and they seem to have a pretty nice/concise tutorial on their library, you can check it out yourself: [Boost.filesystem library tutorial](https://www.boost.org/doc/libs/1_83_0/libs/filesystem/doc/tutorial.html) ([https://www.boost.org/doc/libs/1\\_83\\_0/libs/filesystem/doc/tutorial.html](https://www.boost.org/doc/libs/1_83_0/libs/filesystem/doc/tutorial.html))<sup>18</sup>

👍 0

➡ Reply

## Links

1. <https://www.learncpp.com/author/Alex/>
2. [https://en.cppreference.com/w/cpp/io/basic\\_filebuf/open](https://en.cppreference.com/w/cpp/io/basic_filebuf/open)
3. <https://www.learncpp.com/cpp-tutorial/random-file-io/>
4. <https://www.learncpp.com/>
5. <https://www.learncpp.com/cpp-tutorial/stream-states-and-input-validation/>
6. <https://www.learncpp.com/basic-file-io/>
7. <https://www.learncpp.com/wordpress/tiga-and-wordpress-25/>
8. <https://gravatar.com/>
9. <https://www.learncpp.com/cpp-tutorial/basic-file-io/#comment-594054>
10. <https://www.learncpp.com/cpp-tutorial/basic-file-io/#comment-594119>
11. <https://www.learncpp.com/cpp-tutorial/basic-file-io/#comment-594221>
12. <https://www.learncpp.com/cpp-tutorial/basic-file-io/#comment-589138>
13. <https://www.learncpp.com/cpp-tutorial/basic-file-io/#comment-588838>
14. <https://www.learncpp.com/cpp-tutorial/basic-file-io/#comment-588895>
15. <https://www.learncpp.com/cpp-tutorial/basic-file-io/#comment-580927>
16. <https://www.learncpp.com/cpp-tutorial/basic-file-io/#comment-588786>
17. <https://en.cppreference.com/w/cpp/filesystem>
18. [https://www.boost.org/doc/libs/1\\_83\\_0/libs/filesystem/doc/tutorial.html](https://www.boost.org/doc/libs/1_83_0/libs/filesystem/doc/tutorial.html)
19. <https://g.ezoic.net/privacy/learncpp.com>

