# File I/O

# Recap

# Reading and Writing Disk Files

You can also read and write files stored on your hard disk:

- plain text files
- binary information (a binary file)
  - Such as images or audio recording

To read/write files, you use *variables* of the stream types:

**`ifstream`** for input from plain text files.
**`ofstream`** for output to plain text files.
**`fstream`** for input and output from binary files.

You must `#include <fstream>`

# Code for opening a stream

```
ifstream in_file;
in_file.open("input.txt"); //filename is input.txt
```

An alternative shorthand syntax combines the 2 statements:

```
ifstream in_file(“input.txt”);
string name;
int number;
in_file >> name >> number;
```

# Closing a Stream

- When the program ends, all streams that you have opened will be automatically closed.
- You *can* manually close a stream with the **close** member function:

```
in_file.close();
```

1. Create variable
2. Open file (provide filename)
3. Check if file opened successfully
4. Read from file
5. Close file

# Reading from a stream

- The **>>** operator returns a “not failed” condition, allowing you to combine an input statement and a test.
- A “failed” read yields a **`false`** and a “not failed” read yields a **`true`**.

```
if (in_file >> name >> number)
{
   // Process input
}
```

# Reading from a stream

- You can even read ALL the data from a file because running out of things to read causes that same “failed state” test to be returned:

```
while (in_file >> name >> number)
{
   // Process input
}
```

# Reading A Whole Line: `getline`

- The function **`getline()`** reads a whole line up to the next '\n', into a C++ string.

- The '\n' is then deleted, and NOT saved into the string.

```
string line;
ifstream in_file("myfile.txt");

getline(in_file, line);
```

# Reading A Whole Line in a Loop: `getline`

- The **`getline`** function, like the others we've seen, returns the "not failed" condition.
- To process a whole file line by line:

```
string line;
while( getline(in_file, line)) //reads whole file
{
   // Process line
}
```

# Writing to Files

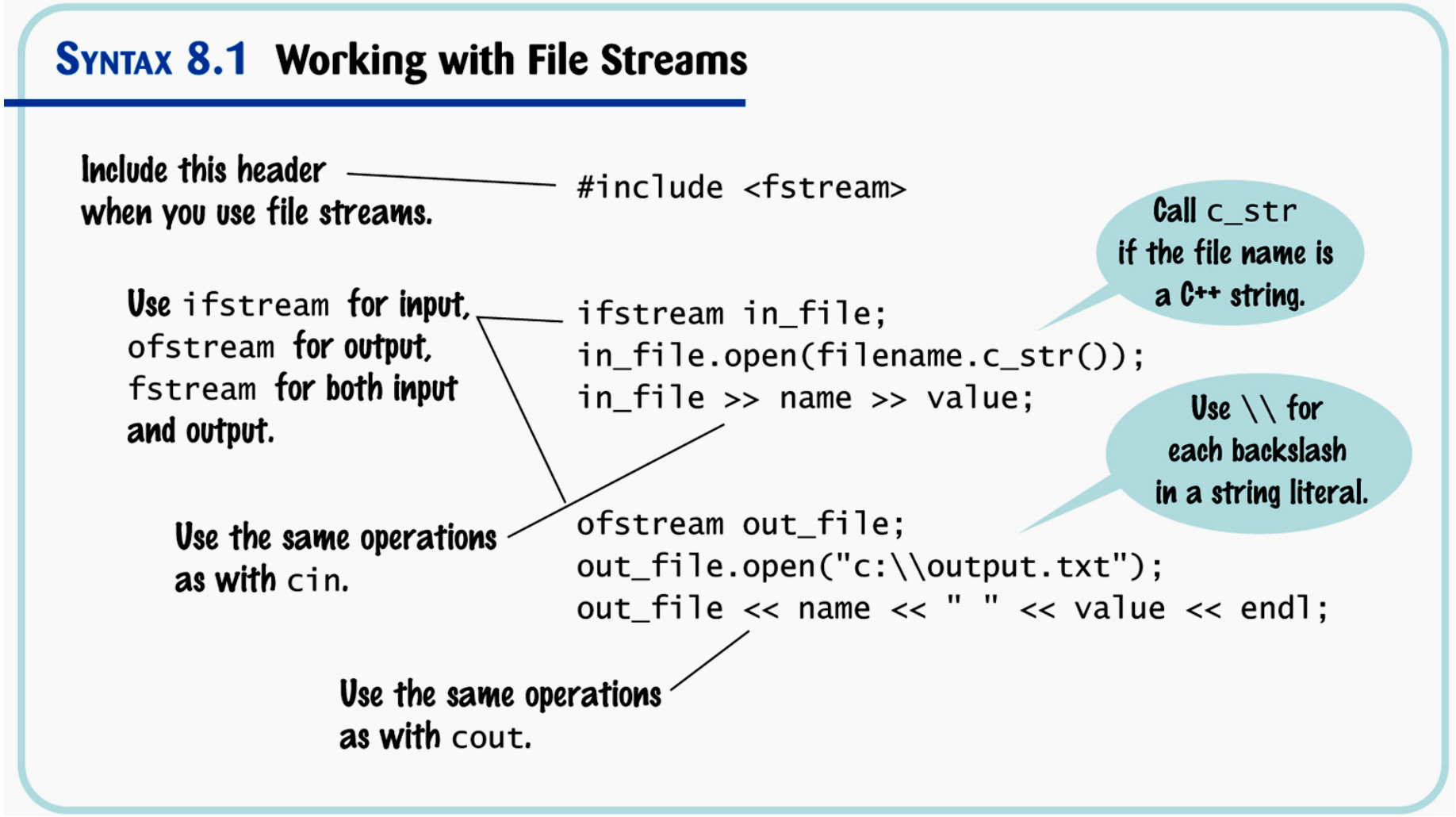
# Writing to a Stream

Here's everything:

1. create output stream variable
2. open the file
3. write to file
4. close file!

```
ofstream out_file;
out_file.open("output.txt");
if (in_file.fail()) { return 0; }
out_file << name << " " << value << endl;
out_file << "CONGRATULATIONS!!!" << endl;
```

# Working with File Streams



# When the File Name is in a C++ string variable

- If the filename comes from the user, you will store it in a string.

- If you use a C++ string, some older library versions require you to convert it to a C-string using `c_str()` as the argument to open():

```
cout << "Please enter the file name:";
string filename;
cin >> filename;
ifstream in_file;
in_file.open(filename.c_str());
```

# Opening a Stream, Filename is a `char[]` array

```
cout << "Please enter the file name:";
char filename[80];
cin >> filename;
ifstream in_file;
in_file.open(filename);
```