Files and Streams II

Working with Files

When using files, you should always follow the same basic steps:

- open the file
- check for failure
- read or write some data
- close the file

For reading data:

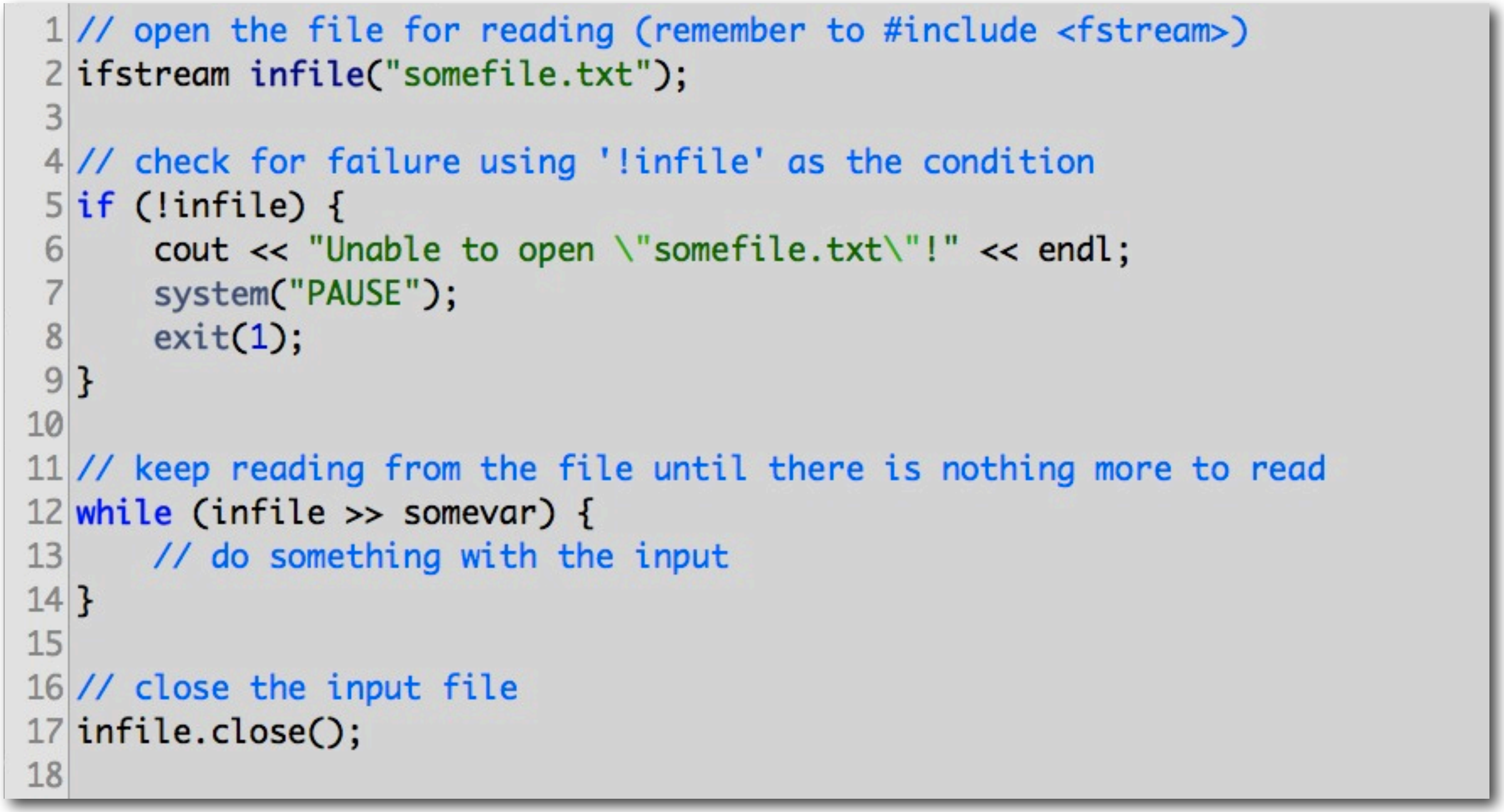
- use ifstream

For writing data:

- use ofstream

Reading from a File

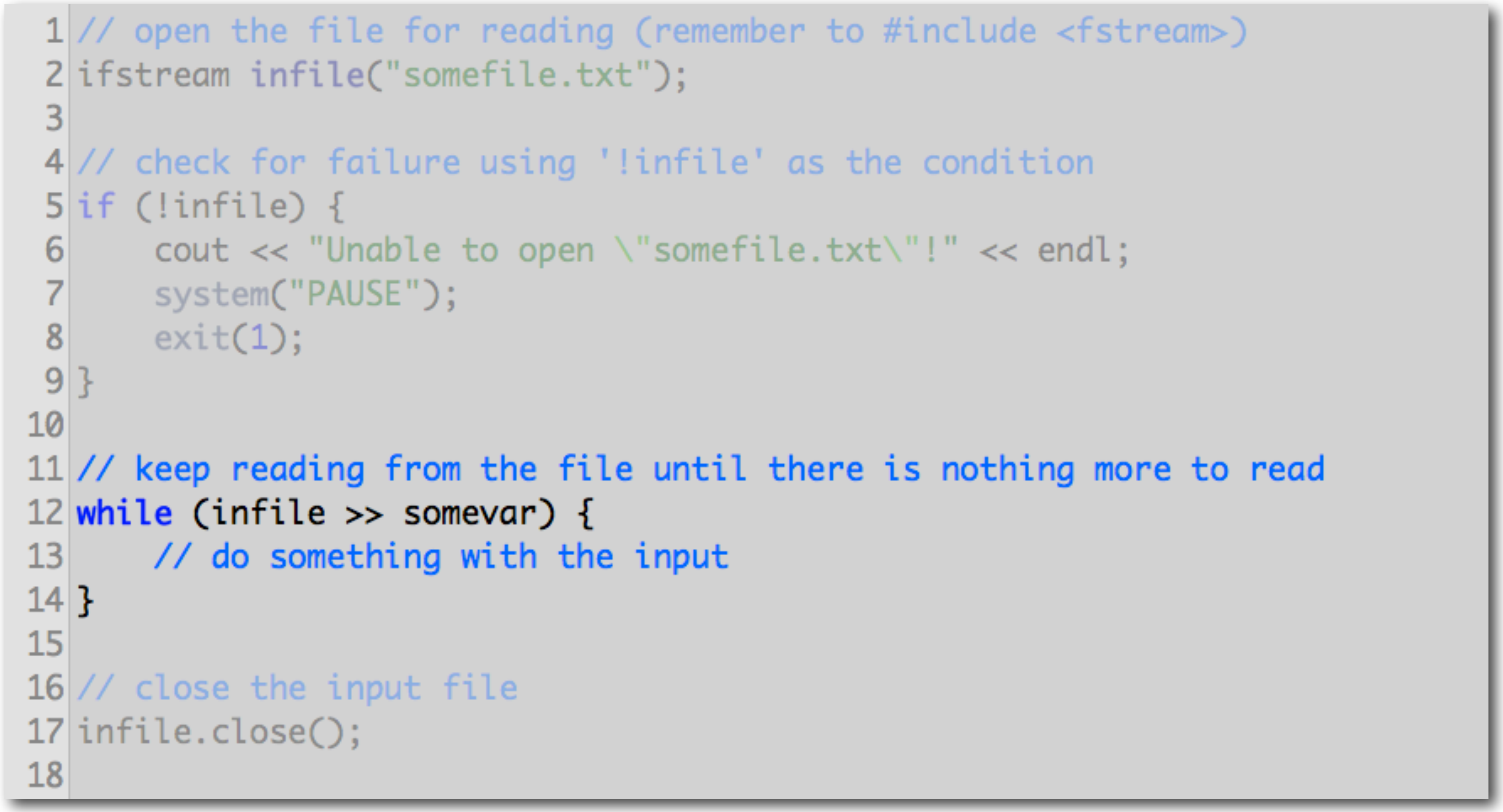
General template:



```
// open the file for reading (remember to #include <fstream>)
ifstream infile("somefile.txt");

// check for failure using '!infile' as the condition
if (!infile) {
    cout << "Unable to open \"somefile.txt\"!" << endl;
    system("PAUSE");
    exit(1);
}

// keep reading from the file until there is nothing more to read
while (infile >> somevar) {
    // do something with the input
}

// close the input file
infile.close();

```

Reading from a File

General template (remember your data will be different):



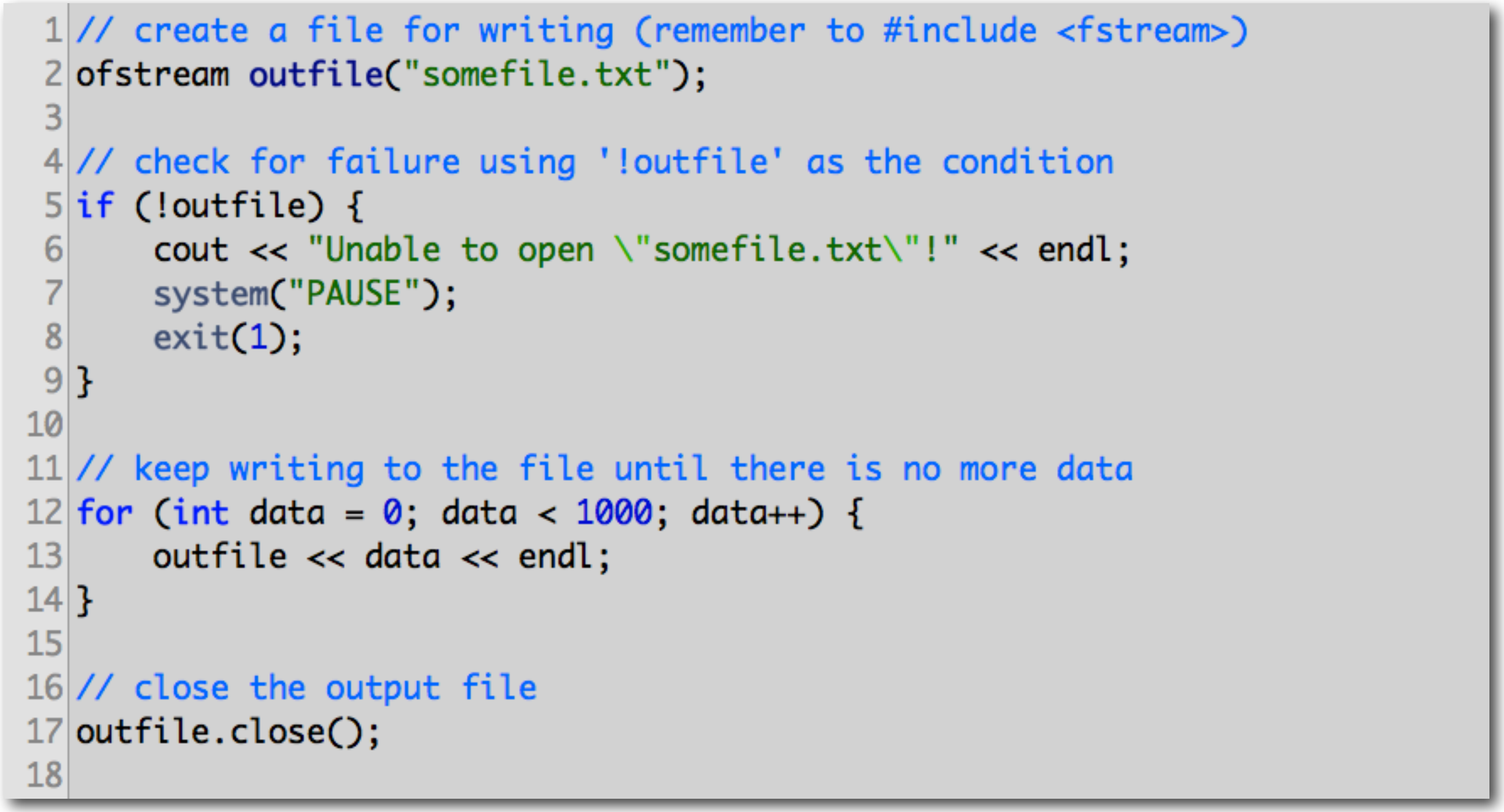
```
// open the file for reading (remember to #include <fstream>)
ifstream infile("somefile.txt");

// check for failure using '!infile' as the condition
if (!infile) {
    cout << "Unable to open \"somefile.txt\"!" << endl;
    system("PAUSE");
    exit(1);
}

// keep reading from the file until there is nothing more to read
while (infile >> somevar) {
    // do something with the input
}

// close the input file
infile.close();

```

Writing to a File

The process for writing to a file is
very similar to that for reading from one!

Writing to a File

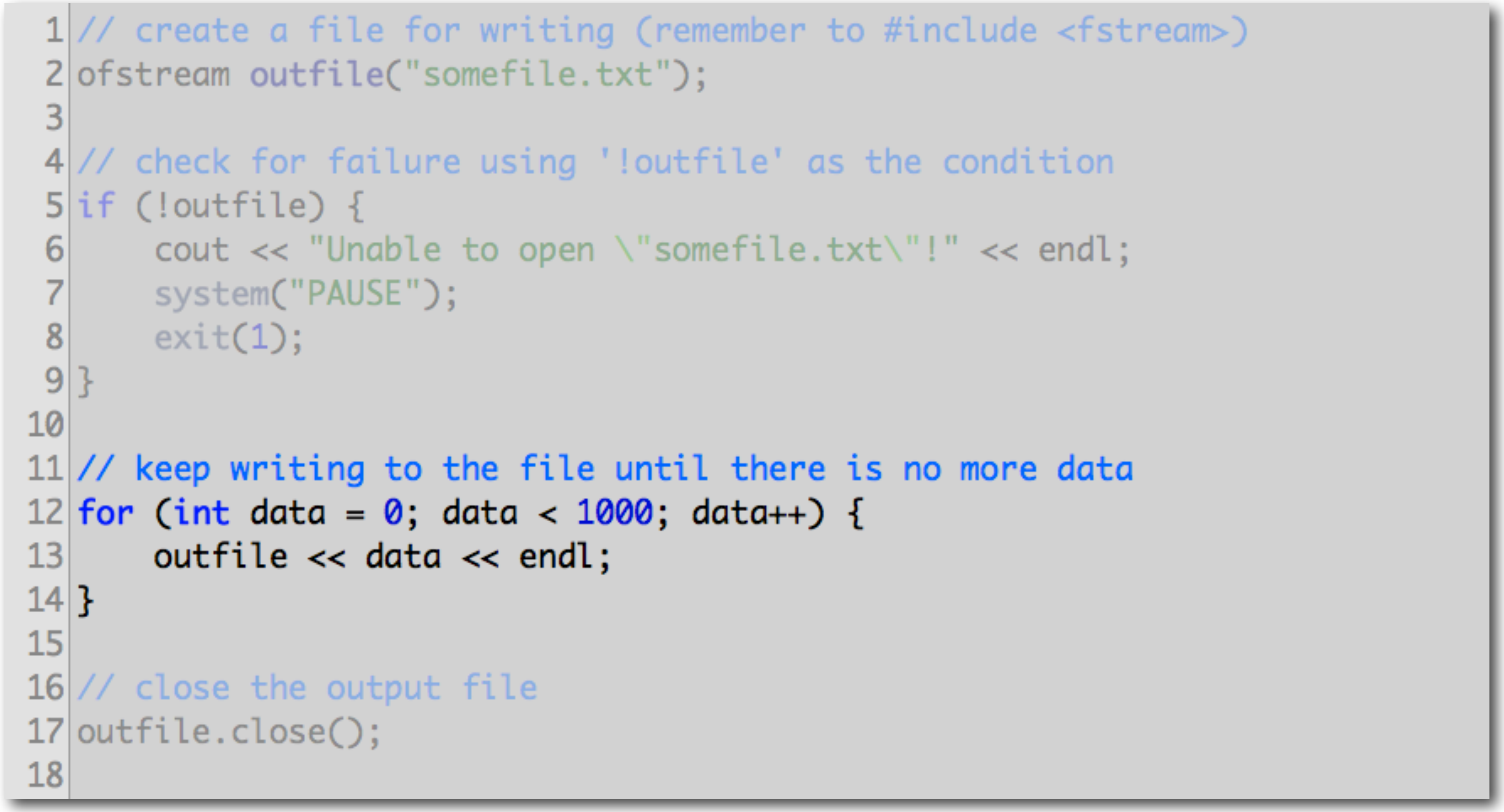
General template:



```cpp
// create a file for writing (remember to #include <fstream>)
ofstream outfile("somefile.txt");

// check for failure using '!outfile' as the condition
if (!outfile) {
    cout << "Unable to open \"somefile.txt\"!" << endl;
    system("PAUSE");
    exit(1);
}

// keep writing to the file until there is no more data
for (int data = 0; data < 1000; data++) {
    outfile << data << endl;
}

// close the output file
outfile.close();

```

Writing to a File

General template (remember your data will be different):



```
// create a file for writing (remember to #include <fstream>)
ofstream outfile("somefile.txt");

// check for failure using '!outfile' as the condition
if (!outfile) {
    cout << "Unable to open \"somefile.txt\"!" << endl;
    system("PAUSE");
    exit(1);
}

// keep writing to the file until there is no more data
for (int data = 0; data < 1000; data++) {
    outfile << data << endl;
}

// close the output file
outfile.close();

```

Opening a File for Writing

General syntax:

```
// remember to #include <fstream>
ofstream identifier("somefile.txt");
```

This creates an ofstream object

- basically just a variable whose data type is ofstream
- identifier is the name you'll use to refer to the stream
- you *must* use the parenthesis to specify the file to open (not the assignment operator)
- the file will be <u>created</u> in the same directory as your project (or <u>overwritten</u> if it already exists!)

Opening a File for Writing

For this class, if the file didn't open successfully, just print an error message and exit the program. Nice and simple. =)

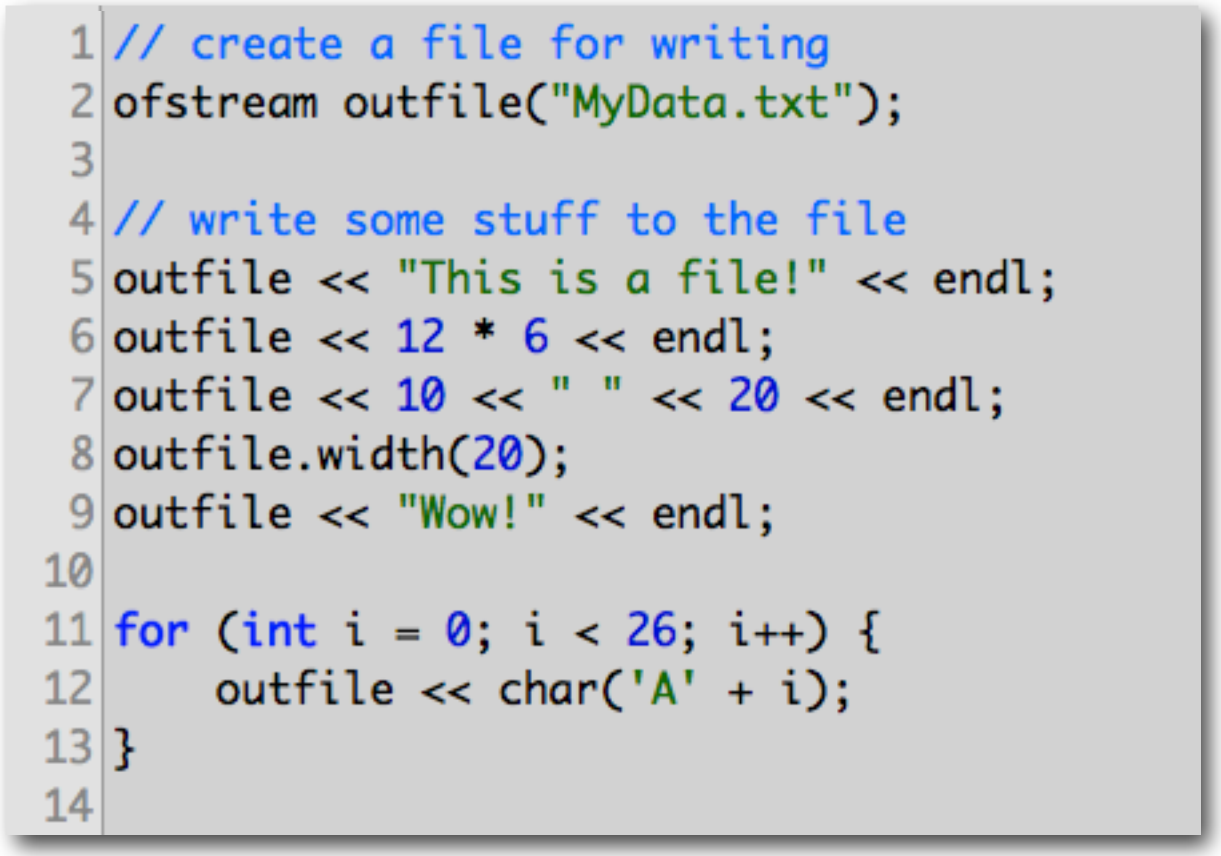
```
   // create the ofstream object
    ofstream outfile("myfile.dat");

    // check for failure
    if (!outfile) {
        cout << "Unable to open the file!" << endl;
        exit(1);         // end the program
    }
```

Writing to a File

You can use an ofstream object much like you would cout

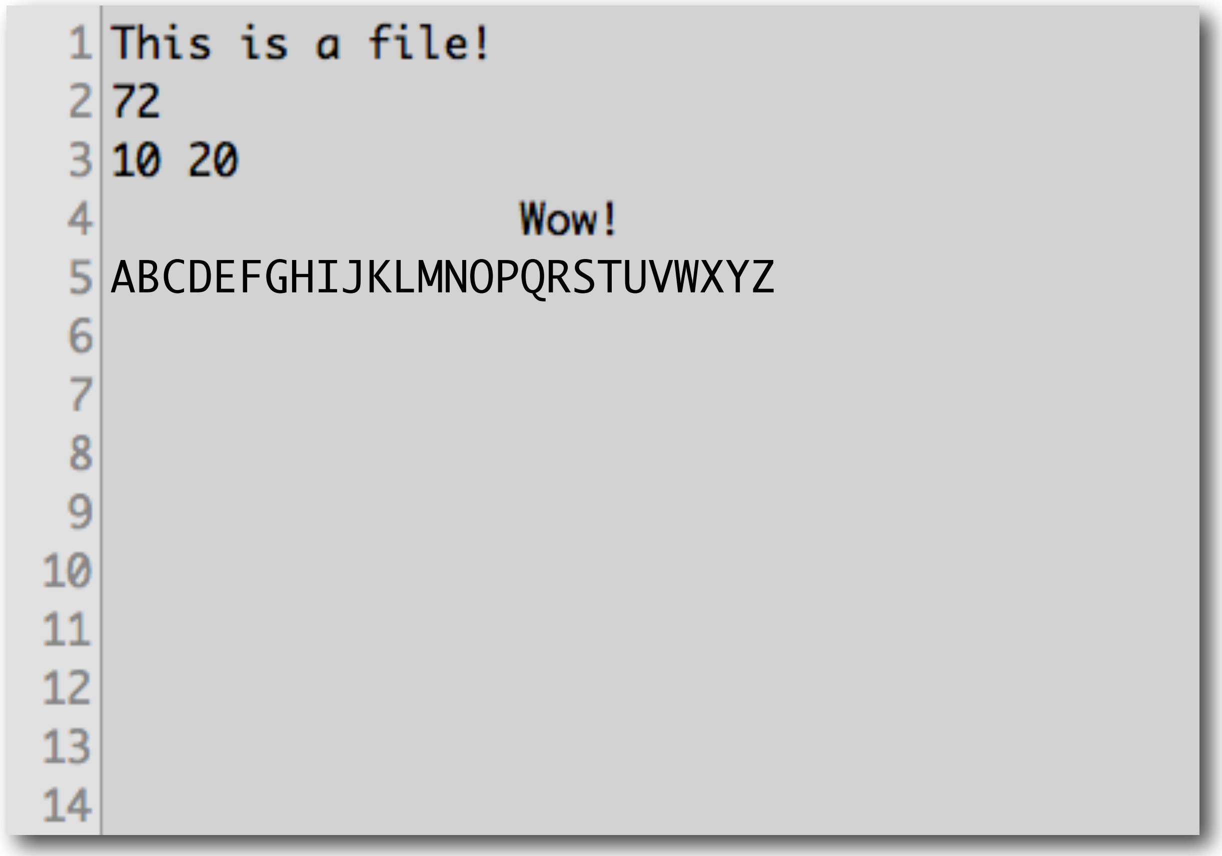
- both use the << operator
- um… it’s really easy?

MyProgram.cpp



```cpp
// create a file for writing
ofstream outfile("MyData.txt");

// write some stuff to the file
outfile << "This is a file!" << endl;
outfile << 12 * 6 << endl;
outfile << 10 << " " << 20 << endl;
outfile.width(20);
outfile << "Wow!" << endl;

for (int i = 0; i < 26; i++) {
    outfile << char('A' + i);
}
```

MyData.txt



```
This is a file!
72
10 20
                Wow!
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Closing Files

After you're done using your files, you should explicitly close them:

```
// create the file stream
ofstream outfile("myfile.dat");

// check for failure and then write some data...

// close the file stream
outfile.close();
```

`close()` is a member function of filestream objects

Writing to a File

Run the `integerOutputDemo.cpp` demo. =)

getline()

General syntax:

```
// extracts text from ‘stream’ until ‘endChar’ is
// encountered and stores the result into ‘str’
getline(stream, str, endChar);
```

What the function does:

- reads from the specified input stream (`stream`) until some delimiting character (`endChar`) is found.
- the resulting text is stored into the specified string variable (`str`)
- the text will NOT include the delimiting character (`endChar`), though the file pointer will be advanced past it

getline()

Alternate syntax:

```
    // extracts text from ‘stream’ until ‘\n’ is
    // encountered and stores the result into ‘str’
    getline(stream, str);
```

If you omit the delimiting character:

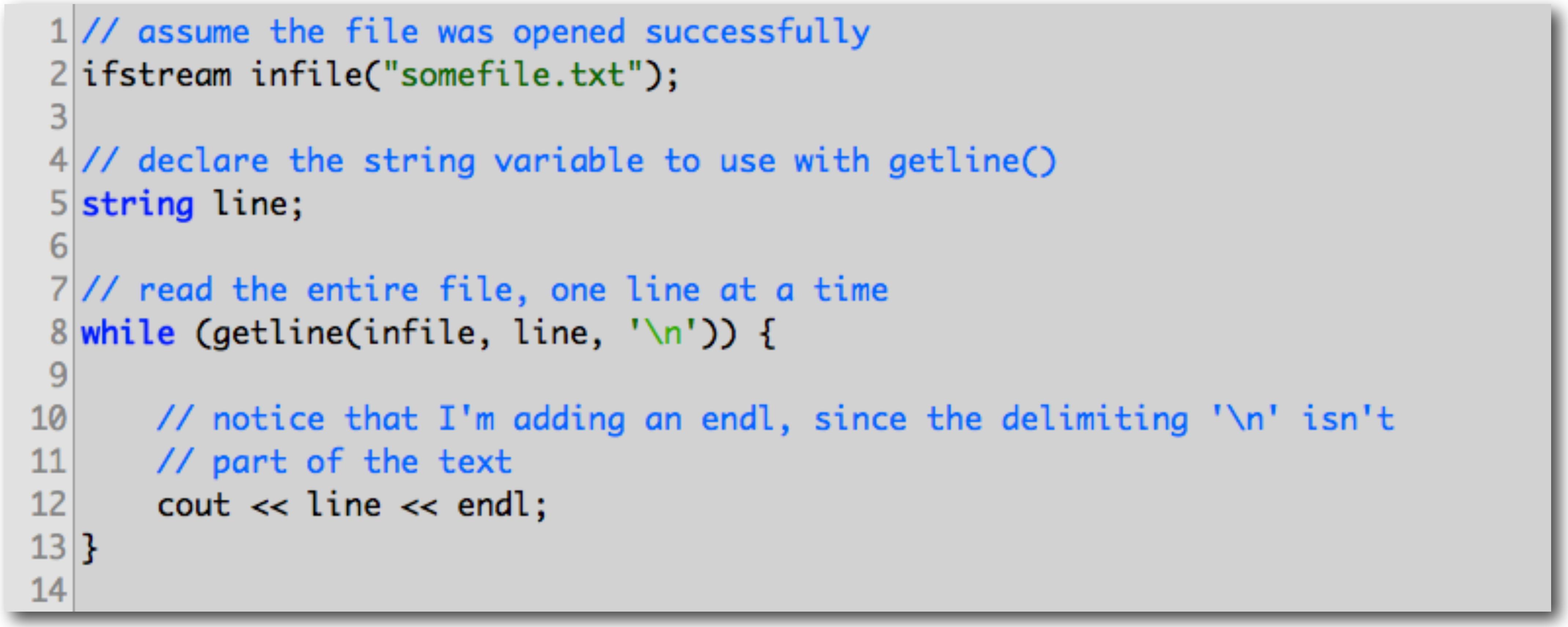
- the function will use the newline character (‘`\n`’) as the default
- this gets an entire line of text from the stream, just like the function name implies

getline()

getline() returns the stream object

- hopefully you remember that stream objects can be "evaluated" as true or false

So, we can use getline() as the condition for a loop:



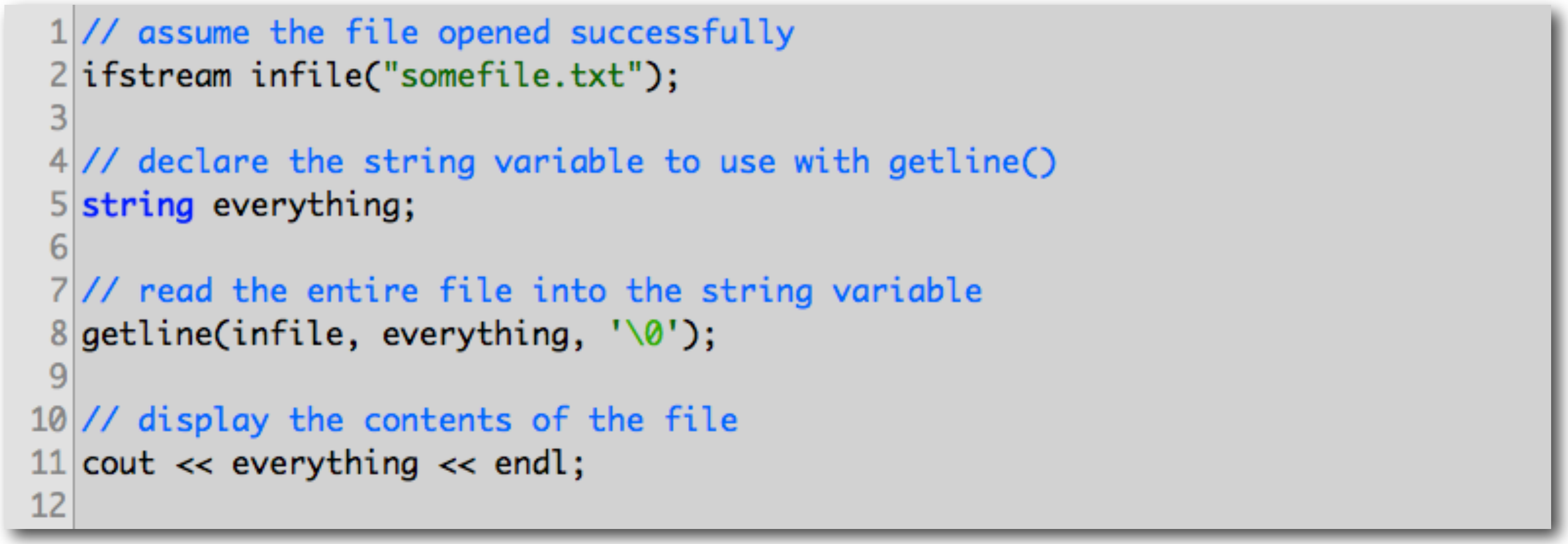
```cpp
// assume the file was opened successfully
ifstream infile("somefile.txt");

// declare the string variable to use with getline()
string line;

// read the entire file, one line at a time
while (getline(infile, line, '\n')) {

    // notice that I'm adding an endl, since the delimiting '\n' isn't
    // part of the text
    cout << line << endl;
}

```

getline()

getline() can be used to read an entire file into a string all at once

- just specify the delimiting character as one you know doesn't occur in the file (such as the null character, '\0')
- no loops required!

Example:



```cpp
// assume the file opened successfully
ifstream infile("somefile.txt");

// declare the string variable to use with getline()
string everything;

// read the entire file into the string variable
getline(infile, everything, '\0');

// display the contents of the file
cout << everything << endl;

```

`getline()`

Run the `getline.cpp` demo. =)