CSCI1300 - Introduction to Computer Programming
Instructor: Dr Knox

Recitation 8: File IO

During this class so far, we have been using the iostream standard library. This library provided us with methods like, cout and cin. *cin* is for reading from standard input and *cout* is for writing to standard output.

In this recitation we will discuss *file Input/Output*, which will allow you to read and write from a file. In order to use these methods, we will need to use another C++ standard library, **fstream**.

Headers to include:

These headers must be included in your C++ file before you are able to process files.

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

Opening a file:

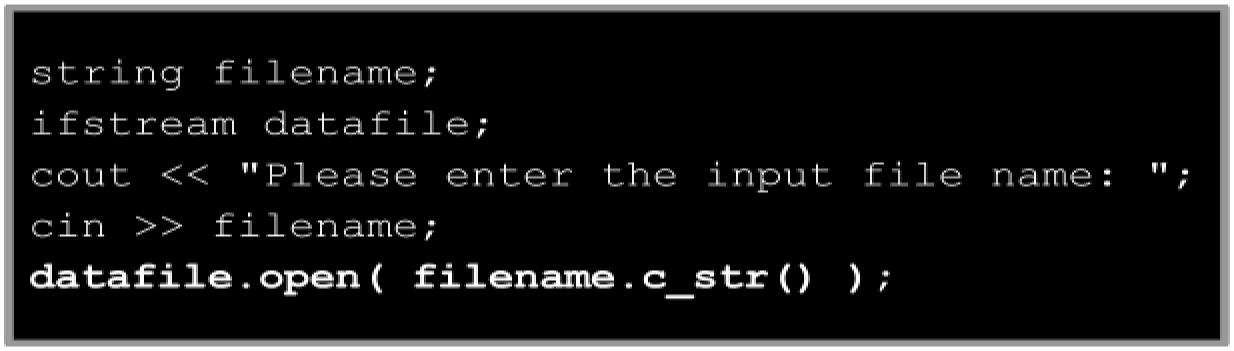
The first step to processing files is to open the file, with ***open(filename)***. It must be opened before you are allowed to read from it or write to it. In order to open a file, the *ofstream*, *fstream*, or *ifstream* objects should be used. If you want to open the file for writing, either the ofstream or fstream object may be used. If you want to open a file for reading only, then the ifstream object should be used.

For example:

```
ofstream myfile;
myfile.open("filename");
```

Variables as file names:

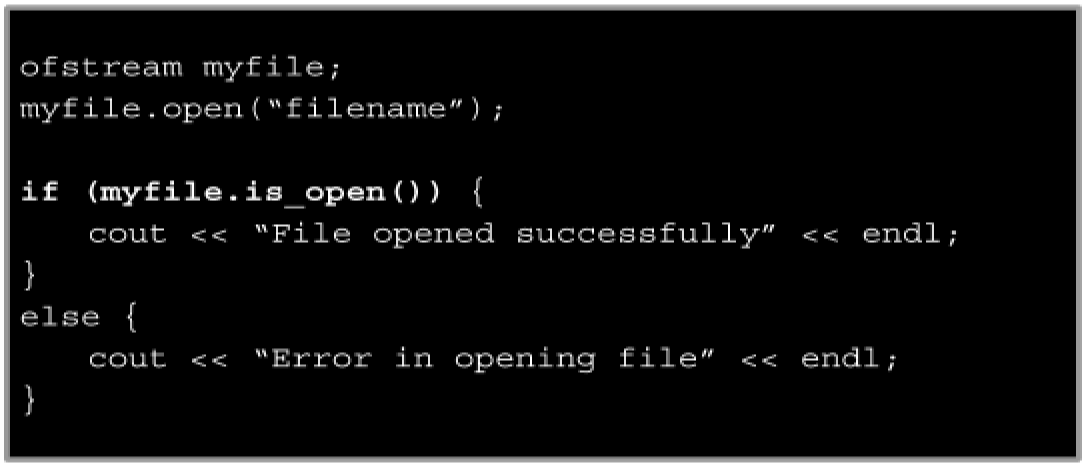
If you are a compiler version less than C++ 11 and you want to store the file name in a string variable first and then use the variable name to open the file, you will have to use the c_str() function. The file name in an open function is a C-style string. Because of specific restrictions in C++, you can't use a variable of type string as the parameter. However, if filename is a variable of type string, you can obtain the corresponding C-style string by calling the function **filename.c_str()**, and you can use this function call as a parameter to open.
Note: For c++ 11 and above, you can directly use the string variable to open the file.
For example:



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

Checking for open file:

It is always good practice to check if the file has been opened properly or send a message that it did not open properly. To check if a file stream successfully opened the file, you can use fileObject**.is_open().** This method will return a boolean value true or false.
You can also use the fileObject.**fail()** function to check if the file open was successful or not.
For example:



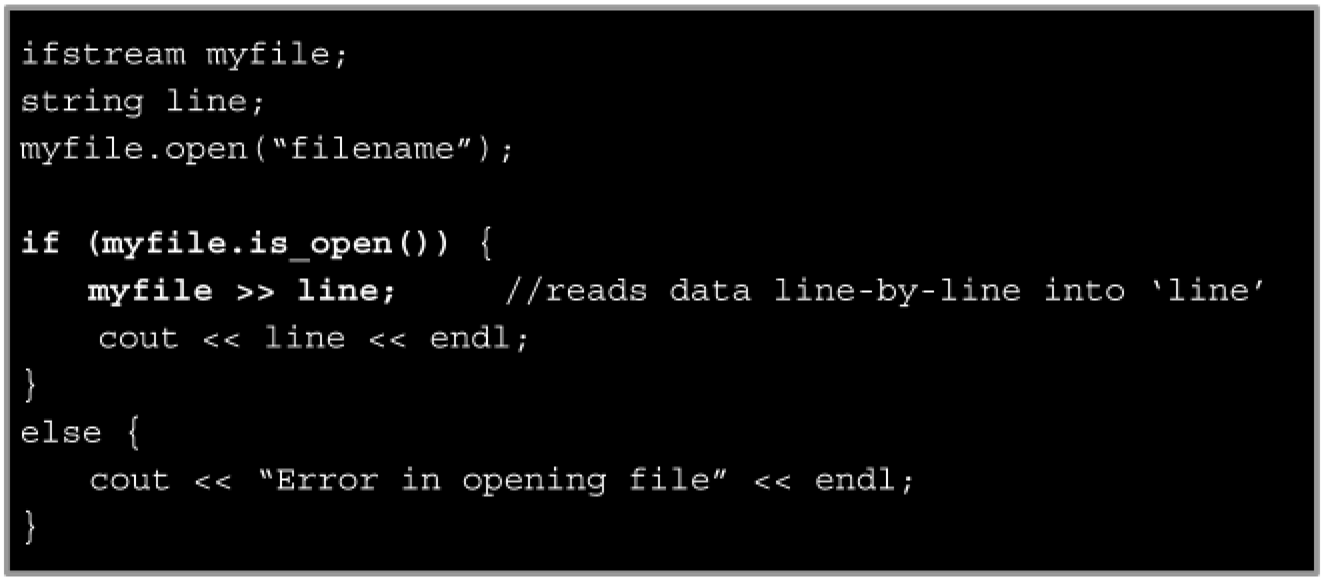
```
ofstream myfile;
myfile.open("filename");

if (myfile.is_open()) {
   cout << "File opened successfully" << endl;
}
else {
   cout << "Error in opening file" << endl;
}
```

Reading from a file:

When you read from a file into your C++ program, you will use the stream extraction syntax you have seen with cin, which inputs information from the keyboard or user, **(>>)**. The difference is that you will be using the ifstream object instead of the cin object, allowing for the program to input or read from the file. Here you would be given a file with information to read. Then once the print statement is called, you will see the lines of the file printed to the terminal.

For example:



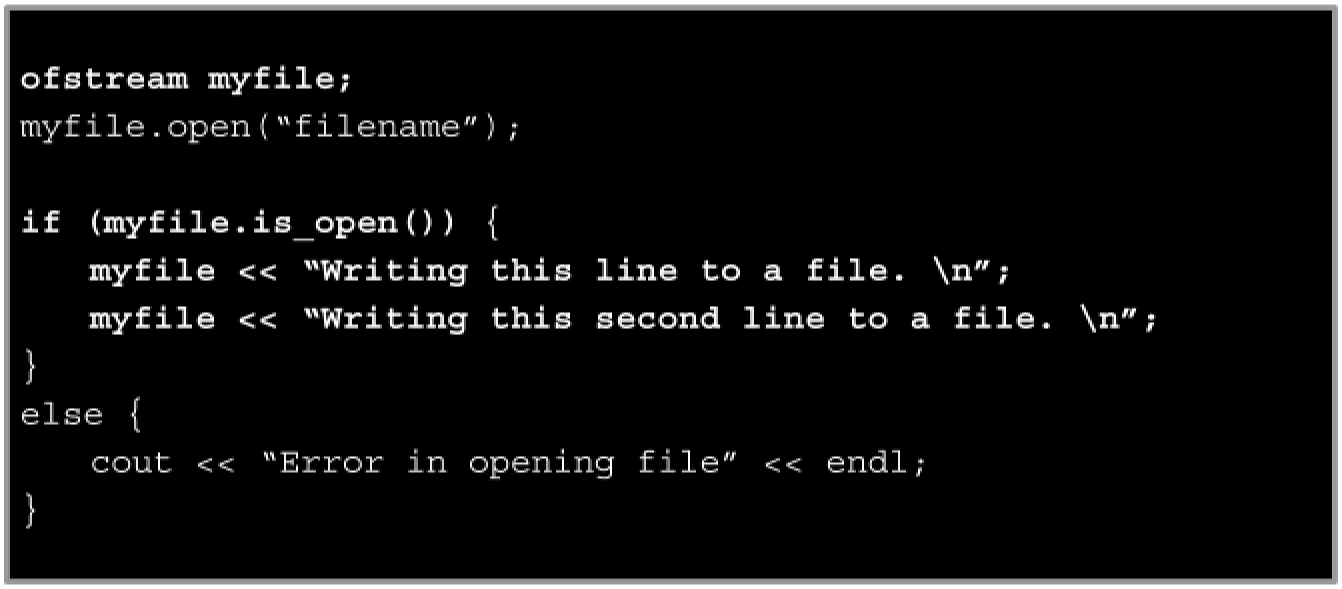
```
ifstream myfile;
string line;
myfile.open("filename");

if (myfile.is_open()) {
   myfile >> line;     //reads data line-by-line into 'line'
    cout << line << endl;
}
else {
   cout << "Error in opening file" << endl;
}
```

Writing to a file:

When you write to a file from your C++ program, you will use the stream insertion syntax you have seen with cout, which outputs information to the terminal screen, **(<<)**. The difference is that you will be using the ofstream object instead of the cout object, allowing for the program to direct the output correctly.

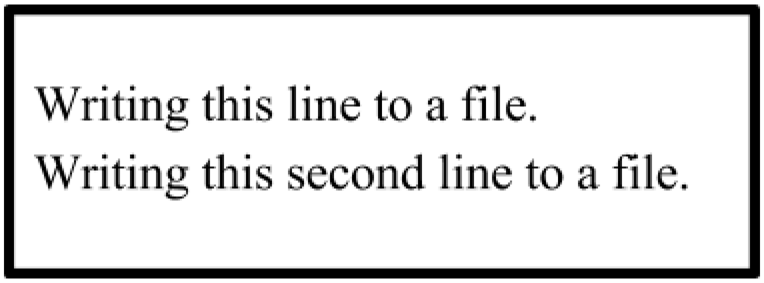
For example:



```
ofstream myfile;
myfile.open("filename");

if (myfile.is_open()) {
   myfile << "Writing this line to a file. \n";
   myfile << "Writing this second line to a file. \n";
}
else {
   cout << "Error in opening file" << endl;
}
```

If you opened the text file, then you will see the string statements within the new output text file, such as:



```
Writing this line to a file.
Writing this second line to a file.
```

Appending to a File:

When we open a file for writing, C++ will assume that we want to **overwrite** the existing contents of the file. In other words, the contents of the file will be deleted and then rewritten. If we want to append new information to an existing file, we need to specify this when the file is opened. We do this by adding the option **fstream::app** as a second argument to the **open** function. This syntax tells the program to look in the included library **fstream** for an option named **app**. The **app** option stands for "append" and tells the function **open** not to delete the given file but to add new information at the end.

For example:

```
ofstream myfile;
myfile.open("filename", fstream::app);
myfile << "Appending a third line... \n";
myfile << "and a fourth. \n";
```

Using the same file “filename” as in the previous example will result in the file contents:

```
Writing this line to a file.
Writing this second line to a file.
Appending a third line…
and a fourth.
```

Closing a file:

When you are finished processing your files, it is always better to close all the opened files before the program is terminated. The standard syntax for closing your file is **close()**.

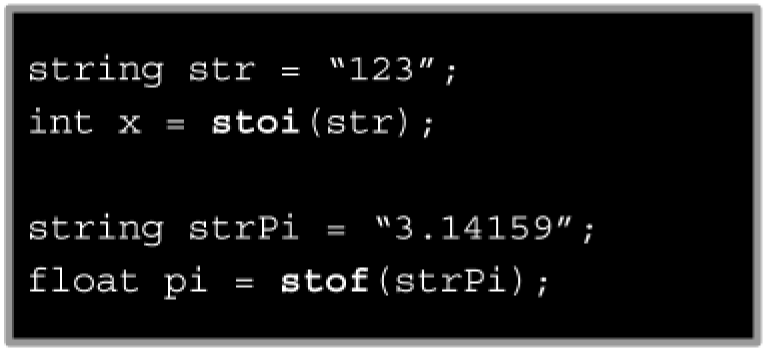
For example:

```
ofstream myfile;
myfile.open("filename");
. . .
myfile.close();
```

Parsing lines of Text using String Streams

Text files are used for storing a wide variety of types of information including strings and numeric values. However, when reading a file the contents are initially interpreted as strings. There are a number of ways of converting strings like “123” and “3.14159” to integers or floats. One way is to use the functions **stoi** (string to int) and **stof** (string to float).

For example:



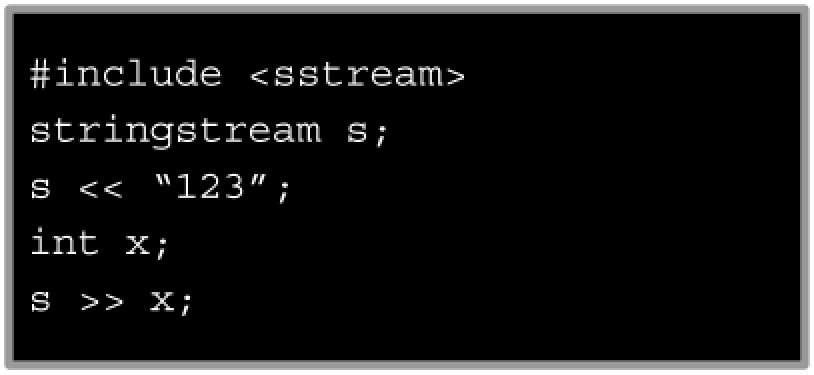
```
string str = “123”;
int x = stoi(str);

string strPi = “3.14159”;
float pi = stof(strPi);
```

Test these functions out on your own. What happens when the string argument to **stoi** or **stof** is not directly representable as an int or float? For instance, what do **stoi(“123.45”)** and **stoi(“123abcd”)** return?

This same task can be accomplished with StringStream. A StringStream acts a lot like a file but exists only within a program. While a file can be saved and accessed after a program has finished running, a StringStream cannot. To use StringStream we must include sstream.

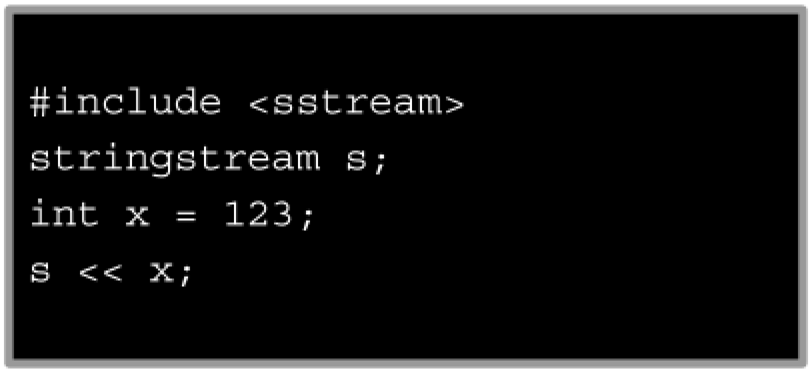
For example:



```
#include <sstream>
stringstream s;
s << “123”;
int x;
s >> x;
```

We can also use StringStream to convert integers and floats to strings.

For example:

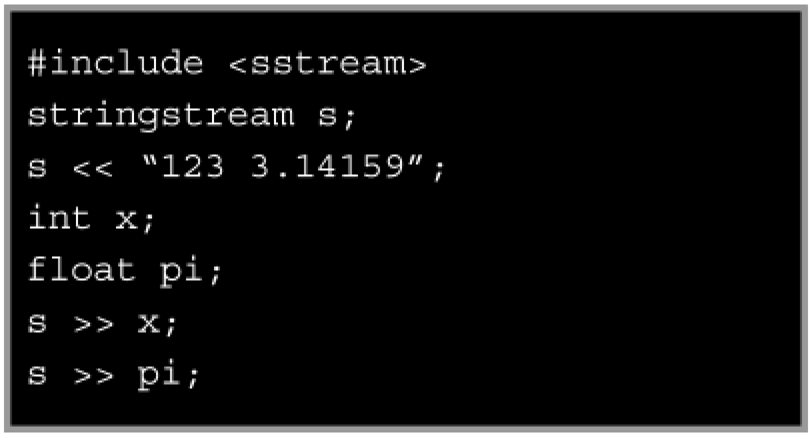


```
#include <sstream>
stringstream s;
int x = 123;
s << x;
```

Data within files can be organized in many ways. Frequently consecutive values on the same line are separated by some specific character called a delimiter. In CSV (Comma Separated Values) files the delimiter is a comma. In TSV (Tab Separated Values) the delimiter is a tab (the escape character for a tab is ‘\t’). When these kinds of files we are faced with another problem. How can the comma or tab separated values on a given line be extracted?

As it turns out, StringStream automatically(by default) uses spaces as a delimiter.

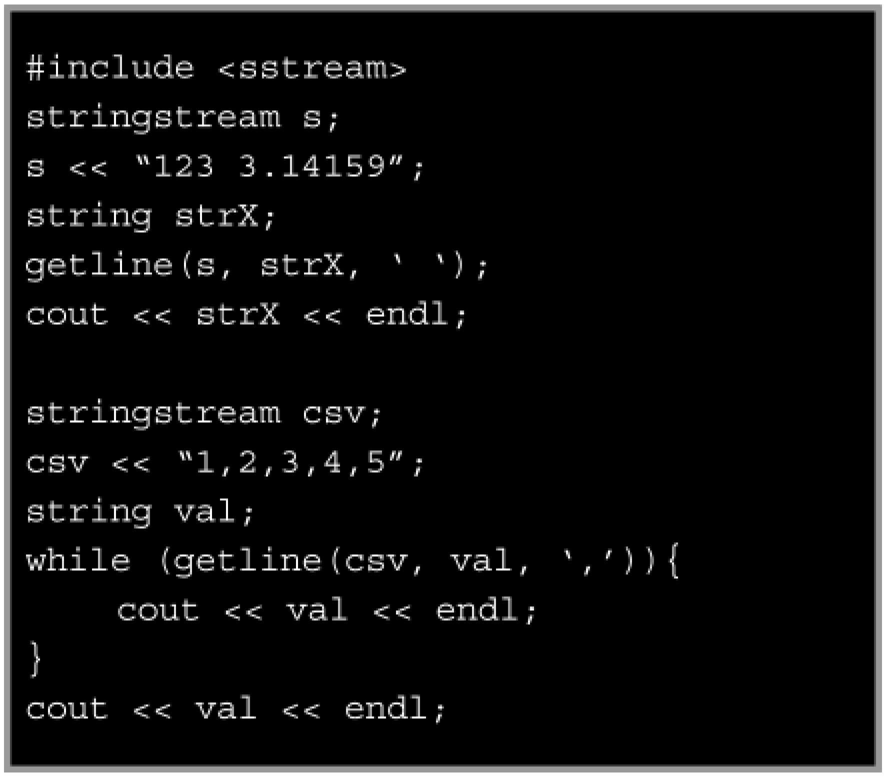
For example:



```
#include <sstream>
stringstream s;
s << “123 3.14159”;
int x;
float pi;
s >> x;
s >> pi;
```

We can specify a delimiter by using the **getline** function. **getline** takes three arguments. The first is some kind of file like object. By “file like” we mean cin, a StringStream, or an fstream object. The second argument is a string variable in which the **getline** will store its result. The last argument is a delimiter character.

For example:



```
#include <sstream>
stringstream s;
s << “123 3.14159”;
string strX;
getline(s, strX, ‘ ‘);
cout << strX << endl;

stringstream csv;
csv << “1,2,3,4,5”;
string val;
while (getline(csv, val, ‘,’)){
    cout << val << endl;
}
cout << val << endl;
```

The code above prints the output:

```
123
1
2
3
4
5
5
```