A Stream Runs Through It

CS 211

Adam Sweeney

November 5, 2017

Wichita State University, EECS

Introduction

- We've been using `cout` and `cin` from the beginning
- Now it's time to gain a better understanding of how they work

Agenda

- Section 6.1
- File I/O
- (Re)Introduction to classes and objects
- Techniques for file I/O

Streams and Basic File I/O

Streams and Basic File I/O

- "As a leaf is carried by a stream, whether the stream ends in a lake or in the sea, so too is the output of your program carried by a stream not knowing if the stream goes to the screen or to a file"
  –Washroom wall of a Computer Science Department (1995)
- As the name implies, a stream is a flow of data
- The data is either flowing into or out of our program
- When the stream flows into our program, it is an input stream
- Conversely, an output stream carries data out of our program
- Streams alone all behave similarly
- It is the source and/or destination that differentiates them

Why Files, Though?

- So far, all the data we've processed, collected, and output has been temporary
- It does not exist once the program is done executing
- Believe it or not, there are times where we want to save the data we are working on
- Files can also greatly simplify the act of inputting a lot of data
  - There's a very good reason our homeworks rarely ask for more than 10 inputs
  - It would be far too tedious for the graders if they had to input large data sets
- It can also help us speed up our own program testing by allowing us to automate inputs

File I/O

- When we take input from a file we are *reading* from the file
- When we stick output in a file we are *writing* to the file
- In learning about file I/O, we will just be reading the entire file in one pass, and writing to a file from the beginning (not adding to an existing file)
- Because data is moving to or from the file, a stream is required
- It is up to us to attach a stream to the file

But First, Declaring a Stream?

- Unlike printing to the screen (`cout`) or reading input from the keyboard (`cin`), streams for reading from or to files are not already declared for us
- This means that we have to declare a stream object, and then tell it which file we want to attach it to

```
#include <fstream>     // Required library for file streams

ifstream inFile;       // An input file stream
ofstream outFile;      // An output file stream
```

- We should observe that the streams are declared just like any other variable

About This Declaring

- One thing that is new is that we have to actually declare our file streams
- Unlike `cout` and `cin` (which are actually the weird ones; what other variables are pre-declared?), file streams are not declared for us ahead of time, so we have to take matters into our own hands
- Just like declaring a variable, we have to say what the type is, and then give it a name
- If we want to **read** from a file, we will declare a stream of type `ifstream`, or input file stream
- If we want to **write** to a file, we will declare a stream of type `ofstream`, or output file stream
- Then names we choose are up to us, but we still want to choose smart names aptly describe what we have declared

Attaching the Stream to the File

- There are two ways we can attach the stream to a file

```
// This method attaches the stream at declaration
ifstream inFile("NAME OF FILE AS C-STRING");

// This method attaches the stream after declaration
ifstream inFile;
inFile.open("NAME OF FILE AS C-STRING");
```

- Note that the name of the file must be passed as a C-String
- This means that if your filename is stored in a `string` object, you will have to use the function `.c_str()` to convert the `string` object to a C-String

Using a File Stream

- Now that we have declared our file stream, and attached it to a file, it's time to use it
- How do we do that?
- Turns out we can use our file stream in the exact same way we have been using the `cout` and `cin` streams
- Again, this is because streams themselves are very similar

```
int x, y;
ifstream inFile("input.txt");


inFile >> x;
inFile >> y;
```

- This is only to illustrate how to use a stream, i.e., an input stream is used just like `cin`, which is also an input stream
- More details on file I/O techniques will be discussed in the final section

(Re)Introduction to Classes

(Re)Introduction to Classes

- File streams are not typical variables
- They are objects, which means that they are instantiations of a class
- In the case of an input stream the class is `istream`, and the output stream class is similarly called `ostream`
- A class is a data structure that can contain data and methods (variables and functions)
- An object is an instantiation of a class, i.e., a variable of a class type
- A function that belongs to a class is called a *member function*
- When we attached a file using the following syntax: `inflie.open("input.txt");` , `open()` is a member function of the `istream` class
- We are not able to access these member functions in a traditional sense that we are used to

Accessing Member Functions

- In order to access a member function, we first need to have what is known as a *calling object*
- Remember, objects can be thought of as variables of a class type
- An object is able to invoke the member functions of a class
- This is because a class is a data structure that collects data and methods
- The member functions do not exist outside of the class
- So it should follow that they do exist inside an object, which can be thought of as a variable of a class type
- So, a calling object is simply a declared object that is calling one of the class functions

A Little More About Member Functions

- Because a calling object is required to call a class member function, the member function is only able to work within the scope of the calling object
- For example the function `open()` can only open a file and attach it to the calling object, and not some othe file stream

Techniques for File I/O

Techniques for File I/O

- When using file streams there are some things we need to do as best practices
- And while the actual stream usage is no different than what we've seen so far with `cin` and `cout`, our ovearall implementations will be a bit different

If We Open a File...

- We had better close it when we're done

```
string filename;
ofstream outFile(filename.c_str());

// Write out to file
outFile.close();
```

- Closing a file is simple, we just call the `close()` function
- NOTE: The operating system will close files for us, **only** if the program exits normally
- We should develop two habits in regards to closing files:
  - Whenever we open a file, we should write the line of code to close it
  - We should close the file as soon as we are done with it

Also, If We Open a File...

- We should make sure that we opened it
- Opening a file is not always successful
- It can fail often enough that we need to make sure it is open before attempting to read or write from/to a file that is not opened

```
#include <cstdlib>

ofstream fileOut("data.dat");
if (fileOut.fail()) {
    cout << "Error opening file for output. Exiting...\n";
    exit(1);
}
// Continue with program as normal
fileOut.close();
```

`exit(1)`?

- The code in the previous example uses a new keyword, `exit`
- It is included in the library `<cstdlib>`
- It is similar to `return`, but with a key difference
- `exit()` will exit the entire program, no matter it is invoked
- If we relegate our file opening to another function, a `return` statement would only exit the function, your program would continue to execute
- However, it is rare that we want our programs to continue to execute if we cannot open a file
- `exit()` ensures that the entire program is terminated
- `exit()` takes a single integer as its parameter. A 0 indicates a successful run, while 1 is tradtionally used to denote an error has occured

Prompting for Input

```
cout << "Enter a word: ";
cin >> word;
```

- We are used to prompting users to enter information, and that's something we should be doing
- Just not with file I/O
- If we are opening a file, it is assumed that:
  - We know what the contents are
  - Program does not need user intervention to read from a file
- So, there is no need to have a prompt asking to enter a piece of information if we are reading or writing to a file

Appending to a File

- It was stated earlier that we would deal solely in writing to a file from scratch
- This is the only expectation that assignment will have
- However, it is worth noting the the fstream library does offer quite a bit more flexibility
- If we have an output file that we do not want to erase every time, we are able to append to it

```
string filename;
ofstream fileOut(filename.c_str(), ios::app);
```

- When opening a file, we can specify what mode we want to read/write in

File Modes

- We are able to open a file in binary or plaintext mode, and we are able to append to files
- There are a few more options available, but they will not be covered in this class
- This class will only deal in writing to files from scratch, in plaintext, every time
    - And these are the default settings, so we don't have to learn anything new
- But showing that appending is possible may motivate some to do further reading on the matter

Filename as Input

- It is perfectly reasonable to assume that we could prompt the user to give us a filename to open
- There are a couple things we need to keep in mind
  - Typos can lead to undesired behavior
  - A typo will either exit the program entirely for failing to open the file, or create a new file with a typo'd name
  - The file stream function `open()` takes a C-String as its parameter
  - This means you must either define your filename variable to be a C-String, or remember to use the function `c_str()` if you choose to go with a `string` object

File of Unkown Length

- It can be quite common to need to read all the data from a file
- It can also be quite common that the amount of data in the file is unknown
- There is a way to continue reading until the end of the file, but it takes a bit of work from us

```
ifstream inFile("input.txt");

while ( !(inFile.eof()) ) {
    // Do stuff
}
```

- Note that a lot of other required code was left out (Checking if file opened, closing the file)
- The function `eof()` returns `true` if the end of the file has been reached, and `false` otherwise