COMP 161 - Lecture Notes - 11 - Streaming File I/O

Spring 2014

In these notes we see how to use the streaming file I/O classes.

File Streams

So far we've encountered three streams: *std::cout*, *std::cerr*, and *std::cin*. These provide a pretty straight forward mechanism for doing *I/O* operations to the standard I/O devices. The *fstream*[1] library provides a set of classes for doing the same style of streaming I/O with files. In short, let lets you create objects to stream out and input to and from a file on the computer.

[1] `http://www.cplusplus.com/reference/fstream/`

The *ifstream*[2] lets you establish a file as a streaming input object and thereby lets you read data from a file. The usage of the *ifstream* is exactly the same as *std::cin*, but setting up the stream takes a little more work. Similarly, the *ofstream*[3] lets you use a file in the same fashion as *std::cout*.

[2] `http://www.cplusplus.com/reference/fstream/ifstream/`

[3] `http://www.cplusplus.com/reference/fstream/ofstream/`

Creating Streams

We don't have to setup *std::cout* and *std::cin* because they have already been setup to write and read from the standard output and standard input respectively. File streams need to be told the path to the file they'll be streaming from. To do this, we initialize them with the path as a string. Path strings can be relative or absolute.

```
ofstream outFile("newlog.log");

ifstream inFile("old/yesterdaylog.log");
```

The *ofstream* named *outFile* can be used to write the file named *newlog.log*. That file must be in the current working directory[4]. The *ifstream* named *inFile* can be used to read data from a file named *yesterdaylog.log* which should be found in the *old* subdirectory of the current working directory. We can also declare uninitialized streams and open a file with them after the fact with the *open* method.

[4] do you see why?

```
ofstream outFile;
ifstream inFile;

outFile.open("newlog.log");
ifstream.infile("old/yesterdaylog.log");
```

There are several quirks with initializing file streams that you need to keep in mind. Let's start with ofstreams. If the file you're attempting to stream to does not currently exist, then *the act of writing to it will create that file*. This let's you create new files in your code. If the file does exist, then the default behavior is to *overwrite* the current file's contents. If you want to append to the file, you need to open it differently. If we wanted to append to *newlog.log*, then we'd open it like this:

```
ofstream outFile("newlog.log",std::ofstream::app);

// alternative
ofstream outFile;
outFile.open("newlog.log",std::ofstream::app);
```

Perhaps the biggest gotcha with file output is that all the data you write won't show up in the file on the hard drive until you close the file stream. *The act of closing the file stream causes the changes to the file to be saved to disc*. To close a our *outFile* stream we simply use the *close* method.

```
outFile.close();
```

The biggest problem you can face with opening an ifstream is attempting to open a file that does not exist. In this case, your write operations will fail. To get around this, we can check that the open succeeding before attempting to read the file. The *is_open*[5] class method is a predicate for exactly this state. The simple strategy we see here is to end the procedure if input file open fails.

[5] http://www.cplusplus.com/reference/fstream/ofstream/is_open/

```
ifstream inFile("mydata.dat");

if( !inFile.is_open() ){
  return ;
}
```

Read and Writing

Once your stream is open, basic reading and writing proceed in the same fashion as they do with *cout* and *cin*. The << operator can be used with output streams to write to files and the >> operator can be used with input streams to read from files. The only difference is the I/O stream object being used.

```
ofstream outFile("writeToMe.txt");
ifstream inFile("readFromMe.txt");
```

```
outFile << "This text appears in the file\n";

// get one int
int x(0);
inFile >> x;

// get 3 a string token (everything to next whitespace char)
string t(""), r(""), u("");
inFile >> t >> r >> u;

// get a whole line of text as a string
string s("");
getline(inFile,s);
```

One thing to keep in mind with file input is a sense of where the read pointer[6] is currently located. Be default, it starts at the beginning of the file. Every time you read from the stream, you advance that pointer. It is possible to use *seekg*[7] to move the read pointer around. The trick is that as far as seekg is concerned, positions are specified in bytes not in logical tokens. So if you need to read the 10th word in a file where words are separated by whitespace, it's probably easier to read and ignore the first 9 words, then read and keep the 10th.

[6] like a cursor

[7] `http://www.cplusplus.com/reference/istream/istream/seekg/`