C++ style file I/O

- Rather than reading input from the keyboard (standard input), we can instead choose to read from a file
- Similarly, we can write to a file instead of writing to the screen (standard output)
- The general sequence is to get a filename, attempt to open the file, check it succeeded, perform our I/O, then close the file
- Attempts to open a file can fail for many reasons: it isn't actually a file, it doesn't exist, we don't have appropriate permissions, etc
- Filenames can even include the path to the file, e.g. `csci160/labex5/somedatafile`

The fstream library

- the routines we'll use are in <fstream>
- if we want to read from a file, we'll create an input file stream, which we'll later connect to a file
- if we want to write to a file, we'll create an output file stream, which we'll later connect to a file

```
ifstream infile;  // infile is our input stream variable
ofstream outfile; // outfile is our output stream variable
```

Opening a file

- we can attempt to open a file by using the open method with our file stream variable, and providing a filename

```
// infile is an input stream variable,
// so tries to open “somefile” for reading
infile.open(“somefile”);
// outfile is an output stream, try to open for output
outfile.open(“anotherfile”);
```

- the filename can be a text literal (like above), or it can be stored in a string variable or a char array

examples

A few attempts to open input files (without error checking so far)

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main(int argc, char *argv[])
{
   ifstream infile1, infile2, infile3;

   // try to open from a cmd line arg
   if (argc > 1) {
      infile1.open(argv[1]);
   }
```

```
// try to open from a string
string fname;
cout << “Enter a filename”;
cin >> fname;
infile2.open(fname);

// try to open from a hardcoded name
infile3.open(“someprogram.cpp”);

.......
```

Checking if the open succeeded

- variable.is_open() can be used to check if the stream opened successfully or not, e.g.

```
ifstream infile;
infile.open(“somefile.txt”);
if (!infile.is_open()) {
   cout << “Sorry, could not open that file” << endl;
} else {
   ... opened ok, now we can use it and later close it ...
}
```

Reading from an open (input) file

- if we successfully opened a file for input then we can use many routines much like cin, e.g.

```
infile >> x;  // read from the file into variable x
getline(infile, s);  // read a line into a string
```

- the various input methods keep track of where we are in the file, each read picks up where the last one left off
- we can test for failed reads using .fail, e.g.

```
if (infile.fail()) {
   ......
```

Checking for end of file

- we might hit the end of the file, the eof() method returns true once we've done a read ***AFTER*** the last actual content

```
do {
     string s;
     infile >> s;
     if (!infile.eof()) {
        cout << “read “ << s << endl;
     }
} while (!infile.eof());
```

- if we forget to check for eof then we could keep re-reading the end of the file over and over and ...

Closing a file when done

- When we have finished with an opened file we close it:

```
infile.close();
```

- note that opening, checking, and closing output files works the same as for input files, e.g.

```
outfile.open(“somefilename”);
if (!outfile.is_open()) {
   cout << “Could not open” << endl;
} else {
   ... do stuff then ...
   outfile.close();
}
```

File output

- if an output file has been successfully opened then we can write to it much the same as with cout, e.g.

```
#include <fstream>
#include <iostream>
#include <string>
using namespace std;

int main()
{
   ofstream outfile;
   string s = “somefilename”;
   float f = 3.94;
```

```
   outfile.open(s);
   if (!outfile.is_open()) {
      cout << “Could not open “ << s << endl;
   } else {
      outfile << “Here is my fancy output” << endl;
      outfile << “F is “ << f << endl;
      outfile.close();
   }
}
```