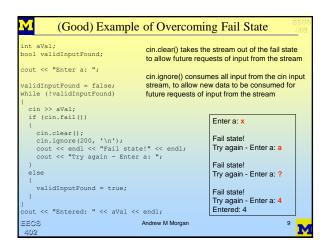
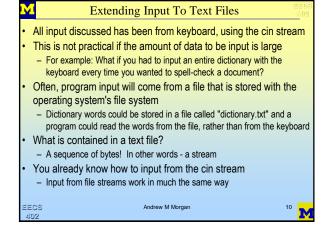
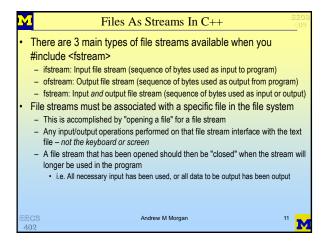


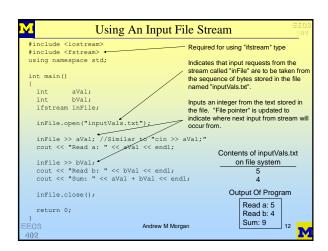
```
(Bad) Example of Overcoming Fail State
int aVal;
bool validInputFound;
                                                            Enter a: x
                                                             Fail state!
cout << "Enter a: ";
                                                             Try again - Enter a:
                                                             Fail state!
                                                             Try again - Enter a:
while (!validInputFound)
                                                             Fail state!
                                                             Try again - Enter a:
  if (cin.fail())
                                                             Fail state!
                                                             Try again - Enter a:
    cout << endl << "Fail state!" << endl;
                                                             Fail state!
    cout << "Try again - Enter a: ";
                                                             Try again - Enter a:
                                                             Fail state!
  else
                                                             .. (infinite loop)
    validInputFound = true;
                                              Note: While it was correctly determined that
                                              the input caused cin to go in the fail state, it
cout << "Entered: " << aVal << endl;
                                             remains in the fail state, and every input
                                              request is ignored, resulting in an infinite loop!
                                   Andrew M Morgan
```

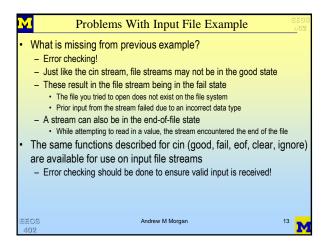
```
(Bad) Example of Overcoming Fail State #2
int aVal;
bool validInputFound;
                                                                               Fail state!
                                                                               Try again - Enter a:
                                                                               Fail state!
                                                                               Try again - Enter a:
validInputFound = false;
while (!validInputFound)
                                                                               Fail state!
                                                                               Try again - Enter a:
   cin >> aVal;
                                                                               Fail state!
                                                                               Try again - Enter a:
                                                                               Fail state!
     cin.clear();
cout << endl << "Fail state!" << endl;
cout << "Try again - Enter a: ";</pre>
                                                                               Try again - Enter a:
                                                                               Fail state!
                                                                              ... (infinite loop)
                                                            Note: cin stream was cleared of the fail state
                                                            but the 'x' that was input remains in the buffer
      validInputFound = true;
                                                            (it is not consumed by the failed input request). As loop iterates it continues trying to consume the 'x' for the value of a, which
cout << "Entered: " << aVal << endl;
                                                            results in the fail state over and over
                                              Andrew M Morgan
```

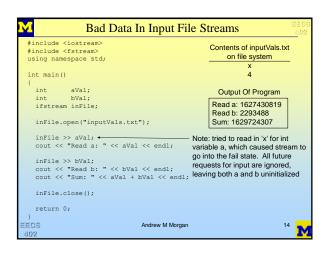




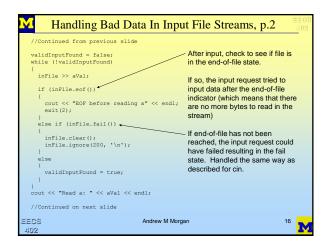




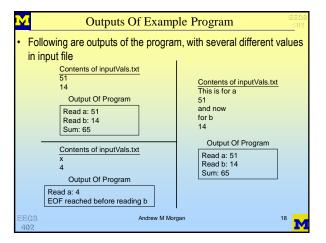




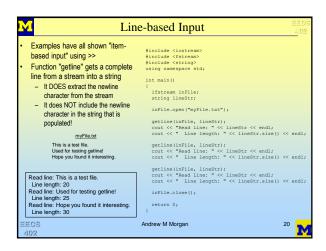
```
Handling Bad Data In Input File Streams, p.1
#include <cstdlib
using namespace std;
int main()
 int
  ifstream inFile;
bool validInputFound;
                                                       Always check to see if opening
the file resulted in a file stream
 inFile.open("inputVals.txt");
 if (inFile.fail())
                                                       in the good state.
   cout << "Unable to open input file!" << endl; If not - the file didn't exist in
   exit(1);
                                                       the file system or the user
                                                       opening the file did not have
 //Continued on next slide
                                                       access to it.
                                    Andrew M Morgan
                                                                                15
```

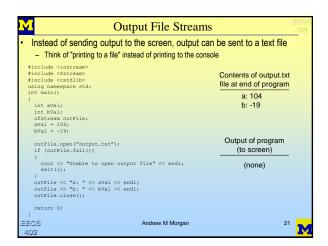


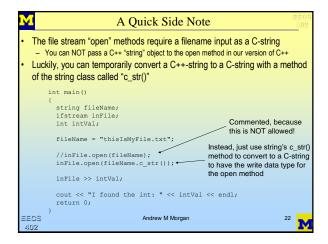
```
Handling Bad Data In Input File Streams, p.3
    alidInputFound = false;
hile (!validInputFound)
     inFile >> bVal;
                                                              Reading in b is done the
                                                              same way as a.
       cout << "EOF before reading b" << endl;
exit(2);</pre>
     else if (inFile.fail())
       inFile.clear();
       inFile.ignore(200, '\n');
     else
       validInputFound = true;
   cout << "Read b: " << bVal << endl;
  cout << "Sum: " << aVal + bVal << endl;
inFile.close();</pre>
  return 0;
ECS
                                     Andrew M Morgan
```



```
Understanding the Stream eof Method
       Remember, the extraction operator (>>) will stop parsing things as soon as it sees the next
       character is a whitespace character
         - Space, Tab, or Newline
   fileWithNewlineAtEnd.txt
                                                               cout << "READING FILE WITH NEWLINE AT END" << endl;
inFile.open("fileWithNewlineAtEnd.txt");
while (!inFile.eof())
                                                                  aWord = "MOTHING HERE";
inFile >> aWord;"
cout << "After word: " << aWord << " EOF: " << inFile.eof() <<
" FAIL: " << inFile.fail() << endl;
                                                               inFile.close();
                                                                 out << endl;
READING FILE WITH NEWLINE AT END
After word: hello EOF: 0 FAIL: 0
After word: here EOF: 0 FAIL: 0
After word: eccs402 EOF: 0 FAIL: 0
After word: class EOF: 0 FAIL: 0
After word: NOTHING, HERE EOF: 1 FAIL: 1
                                                               cout << "READING FILE WITH *NO* NEWLINE AT END" << endl;
inFile.open("fileWithNoNewlineAtEnd.txt");
while (!inFile.eof())
                                                                  aWord = "NOTHING HERE";
inFile >> aWord;
cout << "fer word: " << aWord << " EOF: " << inFile.eof() <<
" FAIL: " << inFile.fail() << end];
READING FILE WITH 'NO' NEW LINE AT END
After word: hello EOF: 0 FAIL: 0
After word: there EOF: 0 FAIL: 0
After word: ees402 EOF: 0 FAIL: 0
After word: class EOF: 1 FAIL: 0
                                                            inFile.close();
                                                                                                                                                                         19
                                                                           Andrew M Morgan
```







```
File Streams are Objects

File streams are objects, and therefore have attributes

One attribute indicates where, in the file, the next read will occur

This means you can pass file stream objects around to functions by reference and have different functions read different parts

In the following example, I show different functions from the same class reading different parts of the data file

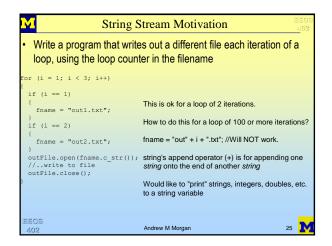
This isn't a requirement though!

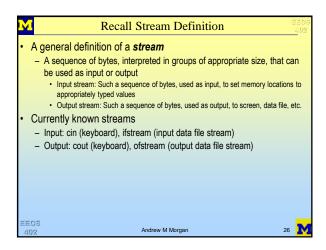
The file stream can be passed to any other function and reading the file will pick up where it left off

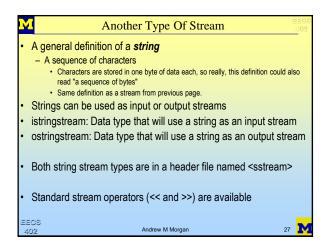
Could be a function of a completely different class!

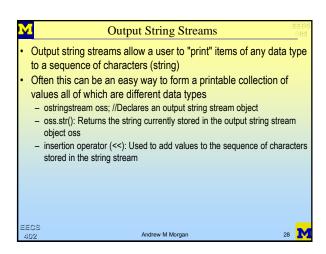
Allows class "A" to read in data from the file that is specific to "A" objects, while still allowing class "B" to read in data specific to class "B"
```

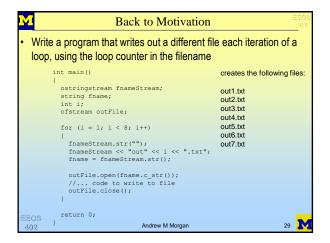
```
Multiple Functions Reading a File, Example
                                                                           public:
  bool readMatrixFile(const string &inFname)
                                                                                bool success;
ifstream inFile;
inFile.open(inFname.c_str());
//error checking here!
success - readHeaderRow(inFile);
//probably more error checking here!
for (int rowInd = 0);
rowInd < MAT_ROWS && success;
rowInd+0.
private:
  int matrixVals[MAT_ROWS][MAT_COLS];
   bool readHeaderRow(ifstream &inFile) const
        inFile >> headerVal;
                                                                                 inFile.close();
      ;
return !inFile.fail();
                                                                       Consider a data file like this:
  bool readValueRow(ifstream &inFile, const int rowInd)
                                                                                  first 2ndCol 3rd Fourth 5th
50 100 150 200 250
87 187 287 387 487
     for (int colInd = 0;
    colInd < MAT_COLS;
    colInd++)</pre>
                                                                                    -11 -22 -33 -44-55
3 1 4 1 5
        inFile >> matrixVals[rowInd][colInd];
                                                          Andrew M Morgan
                                                                                                                                  24
```

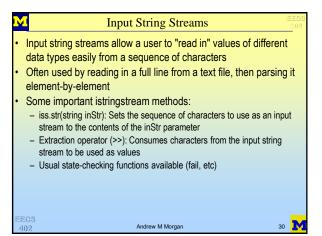












```
M
                              Input String Stream Example
                                                                              printFormat();
                                                                            else
  oid printFormat()
                                                                              if (operStr == "+") {
                                                                                resultVal = lhsVal + rhsVal;
success = true;
                                                                              else if (operStr == "-")
                                                                               resultVal = lhsVal - rhsVal;
success = true;
                                                                              } else //invalid operator! {
                                                                          printFormat();
}
  istringstream iss;
string lineStr;
string varStr;
string eqStr;
double lhsVal;
string operStr;
double resultVal;
bool success = false; //assume not initially
                                                                          cout << "Enter an equation: ";
getline(cin, lineStr);</pre>
                                                                          return 0;
  iss.str(lineStr);
iss >> varStr >> eqStr >> lhsVal >> operStr >> rhsVal;
//continued next column
                                                                    Enter an equation: xVal = 103.6 - 4.7
                                                                    Result: xVal = 98.9
  EECS
                                                                                                                  31
                     Andrew M Morgan
```

