

EECS402 Lecture 10

Andrew M. Morgan

Savitch Ch 12.1-12.2
Streams
Stream States
Input/Output

Andrew M Morgan

1

M

Intro To Streams

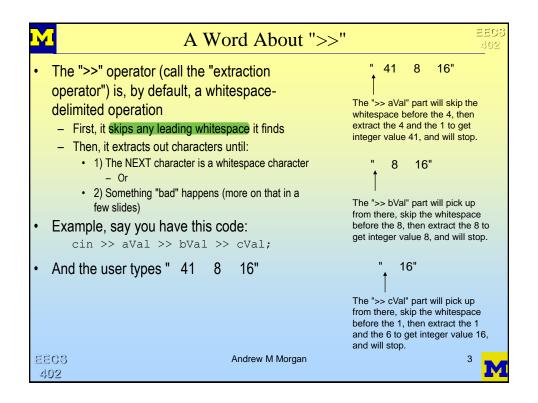
2558 Sent

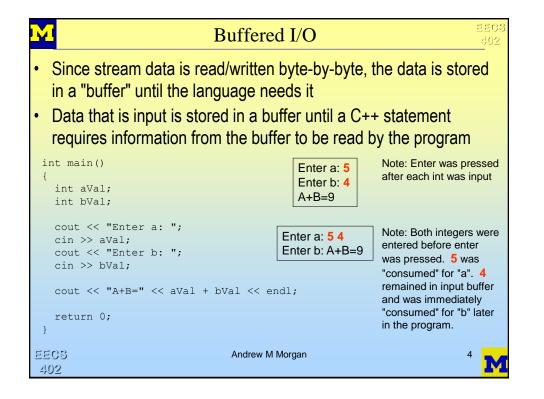
- In C++ a stream is simply "a sequence of bytes"
- · We use a sequence of bytes as input
- Output is also a sequence of bytes
- Therefore, there are two main types of streams
 - Input streams streams that flow from a device in to memory
 - cin is an input stream from the keyboard to memory
 - Output streams streams that flow from memory out to a device
 - · cout is an output stream from memory to the screen
- All I/O is a sequence of bytes, even if multi-byte data types are being used
 - For example, the int value 4,751 is stored in 4 bytes
 - The data is read in byte-by-byte and can be formed into an int with the value 4,751 by C++ (when you use the >> operator, for example)

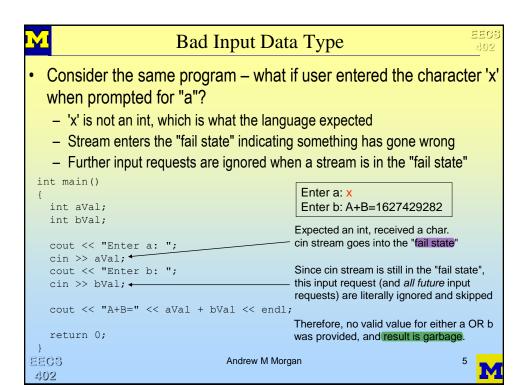
EECS 402 Andrew M Morgan

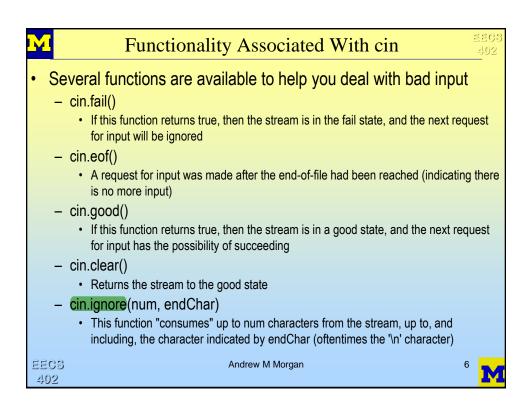
_



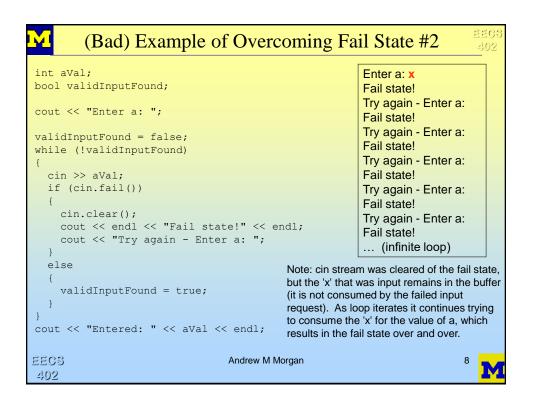








```
(Bad) Example of Overcoming Fail State
int aVal;
bool validInputFound;
                                                              Enter a: x
                                                              Fail state!
cout << "Enter a: ";</pre>
                                                              Try again - Enter a:
                                                              Fail state!
validInputFound = false;
                                                              Try again - Enter a:
while (!validInputFound)
                                                              Fail state!
  cin >> aVal;
                                                              Try again - Enter a:
  if (cin.fail())
                                                              Fail state!
                                                              Try again - Enter a:
     cout << endl << "Fail state!" << endl;</pre>
                                                              Fail state!
     cout << "Try again - Enter a: ";</pre>
                                                              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
                                               remains in the fail state, and every input
cout << "Entered: " << aVal << endl;</pre>
                                               request is ignored, resulting in an infinite loop!
EECS
                                     Andrew M Morgan
703
```



```
(Good) Example of Overcoming Fail State
int aVal;
                                    cin.clear() takes the stream out of the fail state
bool validInputFound;
                                    to allow future requests of input from the stream
cout << "Enter a: ";
                                    cin.ignore() consumes all input from the cin input
                                    stream, to allow new data to be consumed for
validInputFound = false;
while (!validInputFound)
                                    future requests of input from the stream
  cin >> aVal;
  if (cin.fail())
                                                       Enter a: x
    cin.clear();
                                                       Fail state!
    cin.ignore(200, '\n');
    cout << endl << "Fail state!" << endl;</pre>
                                                       Try again - Enter a: a
    cout << "Try again - Enter a: ";</pre>
                                                       Fail state!
 else
                                                       Try again - Enter a: ?
    validInputFound = true;
                                                       Fail state!
                                                       Try again - Enter a: 4
                                                       Entered: 4
cout << "Entered: " << aVal << endl;</pre>
EECS
                                   Andrew M Morgan
                                                                             9
 703
```

M

Extending Input To Text Files

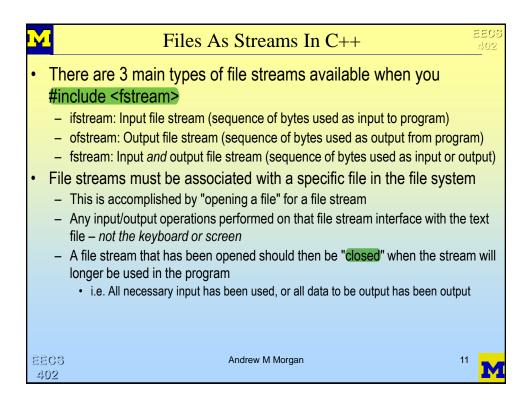
로로C8 402

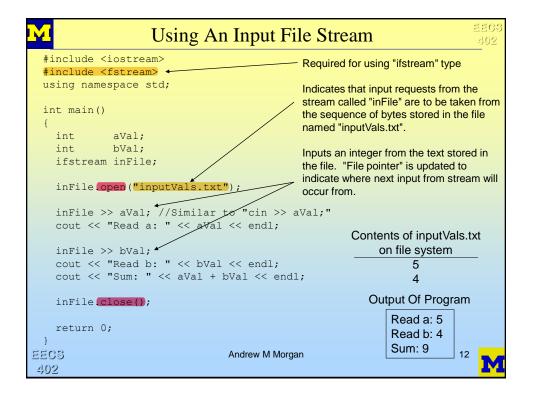
- All input discussed has been from keyboard, using the cin stream
- This is not practical if the amount of data to be input is large
 - For example: What if you had to input an entire dictionary with the keyboard every time you wanted to spell-check a document?
- Often, program input will come from a file that is stored with the operating system's file system
 - Dictionary words could be stored in a file called "dictionary.txt" and a program could read the words from the file, rather than from the keyboard
- What is contained in a text file?
 - A sequence of bytes! In other words a stream
- You already know how to input from the cin stream
 - Input from file streams work in much the same way

402 402 Andrew M Morgan

10









Problems With Input File Example

403 403

- What is missing from previous example?
 - Error checking! super important for project3
 - Just like the cin stream, file streams may not be in the good state
 - These result in the file stream being in the fail state
 - · The file you tried to open does not exist on the file system
 - Prior input from the stream failed due to an incorrect data type
 - A stream can also be in the end-of-file state
 - · While attempting to read in a value, the stream encountered the end of the file
- The same functions described for cin (good, fail, eof, clear, ignore) are available for use on input file streams
 - Error checking should be done to ensure valid input is received!

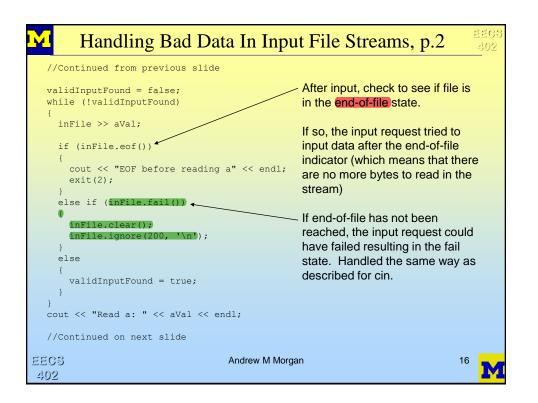
三三CS 402 Andrew M Morgan

13

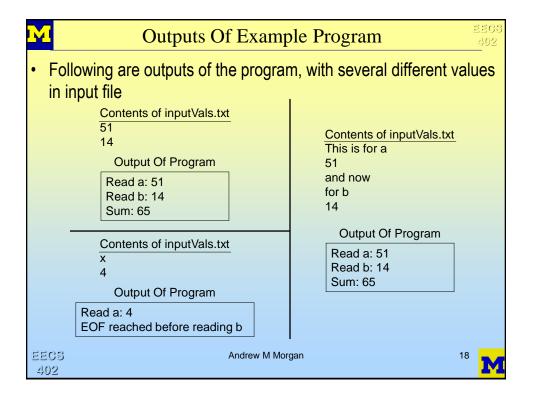


```
Bad Data In Input File Streams
 #include <iostream>
                                                     Contents of inputVals.txt
 #include <fstream>
                                                          on file system
 using namespace std;
                                                                Χ
 int main()
            aVal;
    int
                                                       Output Of Program
            bVal;
                                                      Read a: 1627430819
   ifstream inFile;
                                                      Read b: 2293488
   inFile.open("inputVals.txt");
                                                      Sum: 1629724307
   inFile >> aVal; ←
                                                Note: tried to read in 'x' for int
   cout << "Read a: " << aVal << endl;</pre>
                                                variable a, which caused stream to
                                                go into the fail state. All future
   inFile >> bVal;
                                                requests for input are ignored,
    cout << "Read b: " << bVal << endl;</pre>
                                                leaving both a and b uninitialized
   cout << "Sum: " << aVal + bVal << endl;</pre>
    inFile.close();
    return 0;
EECS
                                  Andrew M Morgan
402
```

```
Handling Bad Data In Input File Streams, p.1
 #include <iostream>
 #include <fstream>
 #include <cstdlib>
 using namespace std;
 int main()
            aVal;
   int.
          bVal;
   int
   ifstream inFile;
           validInputFound;
   bool
                                                     Always check to see if opening
   inFile.open("inputVals.txt");
                                                     the file resulted in a file stream
                                                     in the good state.
   if (inFile.fail())
     cout << "Unable to open input file!" << endl;</pre>
                                                     If not – the file didn't exist in
     exit(1);
                                                     the file system or the user
                                                     opening the file did not have
                                                     access to it.
   //Continued on next slide
EECS
                                   Andrew M Morgan
403
```



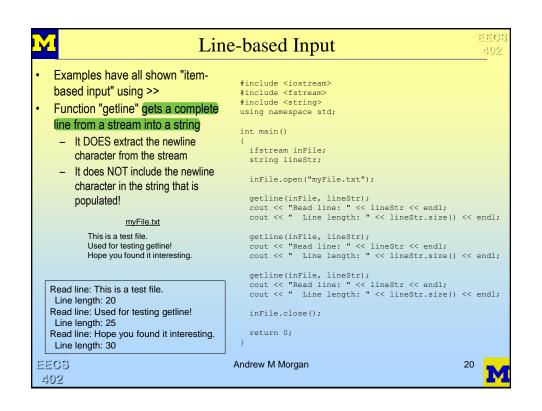
```
Handling Bad Data In Input File Streams, p.3
    validInputFound = false;
    while (!validInputFound)
      inFile >> bVal;
                                                          Reading in b is done the
      if (inFile.eof())
                                                          same way as a.
        cout << "EOF before reading b" << endl;</pre>
        exit(2);
      else if (inFile.fail())
        inFile.clear();
        inFile.ignore(200, '\n');
      else
        validInputFound = true;
    cout << "Read b: " << bVal << endl;</pre>
    cout << "Sum: " << aVal + bVal << endl;</pre>
    inFile.close();
    return 0;
EECS
                                   Andrew M Morgan
703
```



```
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

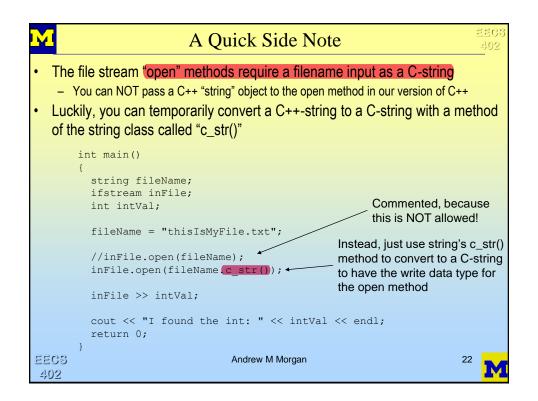
                                         ifstream inFile;
  \underline{fileWithNewlineAtEnd.txt}
                                         string aWord;
     hello there<newline>
                                         cout << "READING FILE WITH NEWLINE AT END" << endl;
     eecs402 class<newline>
                                         inFile.open("fileWithNewlineAtEnd.txt");
                                         while (!inFile.eof())
                                            aWord = "NOTHING HERE":
  fileWithNoNewlineAtEnd.txt
                                            inFile >> aWord;
                                           cout << "After word: " << aWord << " EOF: " << inFile.eof() <<
    " FAIL: " << inFile.fail() << endl;</pre>
     hello there<newline>
     eecs402 class<eof>
                                         inFile.close();
                                         cout << endl;
READING FILE WITH NEWLINE AT END
After word: hello EOF: 0 FAIL: 0
                                         cout << "READING FILE WITH *NO* NEWLINE AT END" << endl;
After word: there EOF: 0 FAIL: 0
                                         inFile.open("fileWithNoNewlineAtEnd.txt");
After word: eecs402 EOF: 0 FAIL: 0
After word: class EOF: 0 FAIL: 0
                                         while (!inFile.eof())
After word: NOTHING_HERE EOF: 1 FAIL: 1
                                            aWord = "NOTHING HERE";
                                            inFile >> aWord;
READING FILE WITH *NO* NEWLINE AT END
                                           cout << "After word: " << aWord << " EOF: " << inFile.eof() <<
    " FAIL: " << inFile.fail() << endl;</pre>
After word: hello EOF: 0 FAIL: 0
After word: there EOF: 0 FAIL: 0
After word: eecs402 EOF: 0 FAIL: 0
After word: class EOF: 1 FAIL: 0
                                         inFile.close();
EECS
                                                   Andrew M Morgan
                                                                                                                 19
 703
```

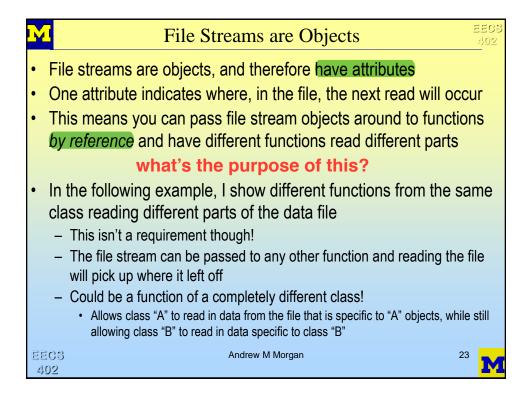


```
Output File Streams
   Instead of sending output to the screen, output can be sent to a text file

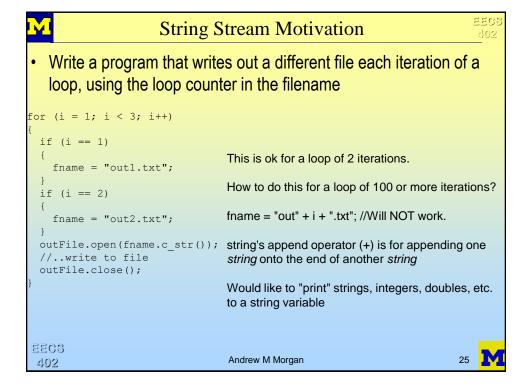
    Think of "printing to a file" instead of printing to the console

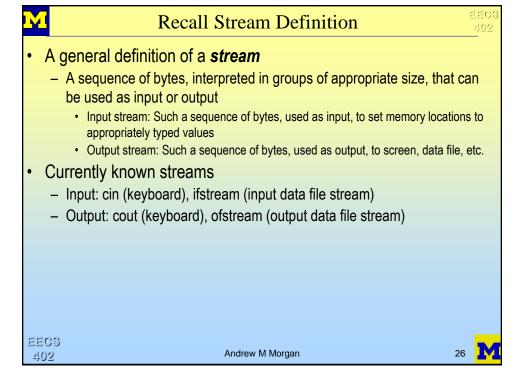
  #include <iostream>
  #include <fstream>
                                                              Contents of output.txt
  #include <cstdlib>
                                                              file at end of program
  using namespace std;
  int main()
                                                                     a: 104
    int aVal;
                                                                     b: -19
    int bVal:
   ofstream outFile;
    aVal = 104;
   bVal = -19;
                                                                Output of program
                                                                    (to screen)
    if (outFile.fail())
     cout << "Unable to open output file" << endl;
                                                                      (none)
     exit(1);
   outFile << "a: " << aVal << endl;
   outFile << "b: " << bVal << endl;
    outFile.close();
    return 0;
EECS
                                      Andrew M Morgan
703
```





```
Multiple Functions Reading a File, Example
const int MAT ROWS = 4;
                                                   bool readMatrixFile(const string &inFname)
const int MAT COLS = 5;
class MatrixInfo
                                                     ifstream inFile;
                                                     inFile.open(inFname.c_str());
   int matrixVals[MAT_ROWS][MAT_COLS];
                                                     //error checking here!
success = readHeaderRow(inFile);
   bool readHeaderRow(ifstream &inFile) const
                                                     //probably more error checking here!
                                                     for (int rowInd = 0;
     string headerVal;
                                                          rowInd < MAT_ROWS && success;
     for (int colInd = 0;
                                                          rowInd++)
          colind < MAT_COLS;
          colInd++)
                                                       success = readValueRow(inFile, rowInd);
                                                       //maybe some more here!
       inFile >> headerVal;
                                                     inFile.close();
     return !inFile.fail();
                        Pass-by reference
                                               Consider a data file like this:
   bool readValueRow(ifstream @inFile,
                                                                                         Header row
                    const int rowInd)
                                                       50 100 150 200 250
     for (int colInd = 0;
                                                       87 187 287 387 487
          colind < MAT COLS;
                                                       -11 -22 -33 -44-55
                                                       3 1 4 1 5
       inFile >> matrixVals[rowInd][colInd];
     return !inFile.fail();
EECS
                                       Andrew M Morgan
402
```







Another Type Of Stream

보트C8 402

- A general definition of a string
 - A sequence of characters
 - Characters are stored in one byte of data each, so really, this definition could also read "a sequence of bytes"
 - · Same definition as a stream from previous page.
- Strings can be used as input or output streams
- istringstream: Data type that will use a string as an input stream
- ostringstream: Data type that will use a string as an output stream
- Both string stream types are in a header file named <sstream>
- Standard stream operators (<< and >>) are available

402 EEC8

Andrew M Morgan

27



M

Output String Streams

402

- Output string streams allow a user to "print" items of any data type to a sequence of characters (string)
- Often this can be an easy way to form a printable collection of values all of which are different data types
 - ostringstream oss; //Declares an output string stream object
 - oss.str(): Returns the string currently stored in the output string stream object oss
 - insertion operator (<<): Used to add values to the sequence of characters stored in the string stream

EECS 402

Andrew M Morgan

M

```
Back to Motivation
 Write a program that writes out a different file each iteration of a
  loop, using the loop counter in the filename
        int main()
                                                      creates the following files:
          ostringstream fnameStream;
                                                      out1.txt
          string fname;
                                                      out2.txt
          int i;
                                                      out3.txt
          ofstream outFile;
                                                      out4.txt
                                                      out5.txt
          for (i = 1; i < 8; i++)
                                                      out6.txt
            fnameStream.str("");
                                                      out7.txt
            fnameStream << "out" << i << ".txt";</pre>
            fname = fnameStream.str();
            outFile.open(fname.c str());
            //... code to write to file
            outFile.close();
          return 0;
EECS
                                Andrew M Morgan
703
```

Input String Streams Input string streams allow a user to "read in" values of different data types easily from a sequence of characters Often used by reading in a full line from a text file, then parsing it element-by-element Some important istringstream methods: iss.str(string inStr): Sets the sequence of characters to use as an input stream to the contents of the inStr parameter Extraction operator (➣): Consumes characters from the input string stream to be used as values Usual state-checking functions available (fail, etc)

```
Input String Stream Example
                                                                             if (iss.fail() || !iss.eof())
 include <iostream>
#include <sstream>
#include <string>
                                                                               printFormat();
#include <string>
using namespace std;
                                                                               if (operStr == "+")
void printFormat()
 cout << "ERROR in format: " <<
    "<var> = <lns> <oper> <rhs>" << endl;
cout << "Example: x = 54.75 + 3.1" << endl;
cout << " Where <lns> and <rhs> are doubles " <<
    "and <oper> can be + -" << endl;</pre>
                                                                                 resultVal = lhsVal + rhsVal;
                                                                                 success = true;
                                                                               else if (operStr == "-")
                                                                                 resultVal = lhsVal - rhsVal;
success = true;
int main()
                                                                               else //invalid operator!
  istringstream iss;
                                                                                 printFormat();
  string lineStr;
  string varStr;
  string eqStr;
  double lhsVal;
                                                                            if (success)
  string operStr;
  double rhsVal;
                                                                               cout << "Result: " << varStr <<
  double resultVal;
                                                                                             " << resultVal << endl;
  bool success = false; //assume not initially
  cout << "Enter an equation: ";</pre>
  getline(cin, lineStr);
  iss.str(lineStr);
  iss >> varStr >> eqStr >> lhsVal >> operStr >> rhsVal;
                                                                       Enter an equation: xVal = 103.6 - 4.7
  //continued next column
                                                                       Result: xVal = 98.9
EECS
                     Andrew M Morgan
 703
```

```
Alternative to atoi and atof
   main(int argc, char *argv[])
 ostringstream oss;
 istringstream iss;
 int intVal;
 double doubleVal;
 char checkChar;
  if (argc != 3)
   exit(2);
                                           [ 57 ] temp -: altForAtoiAtof 🔟
                                                                                   Bad # params
                                           ERROR - wrong number of parameters
                                           Usage: altForAtoiAtof <doubleVal> <intVal>
                                                                                   Good run
 for (i = 1; i < argc; i++)
                                           [ 58 ] temp -: altForAtoiAtof 92.75 16
   oss << " " << argv[i];
                                           92.75 / 16 = 5.79688
                                                                                   Non-parseable
                                           [ 59 ] temp -: altForAtoiAtof hello 18 -
                                           ERROR - invalid format for parameters
                                                                                   values can be
                                           Usage: altForAtoiAtof <doubleVal> <intVal>
                                                                                    seen as error
                                           [ 60 ] temp -: altForAtoiAtof 92.75 16.75
                                           ERROR - invalid format for parameters
 if (iss.fail() || !iss.eof())
                                                                                    "EOF" on stream
                                           Usage: altForAtoiAtof <doubleVal> <intVal>
   cout << "ERROR - invalid format for parameters" << endl;
cout << "Usage: " << argv[0] << " <doubleVal> <intVal>" << endl;</pre>
                                                                                    can indicate
                                                                                    "extra" data
   exit(2):
 cout << doubleVal << " / " << intVal << " = " << (doubleVal / intVal) << endl;
 return 0;
EECS
                                           Andrew M Morgan
```

turn output string stream into input string strean

402

???