File Input/Output

- All input and output streams use:
  - Sequential access
  - The same functions and operators

- Can access multiple streams in the same program
  - cout and cin are just streams we use a lot

- Sample input:

```
This is a set of five numbers
4.5 7 216 0.432 11
```

File Input/Output

- A *file* is an area in secondary storage to hold data

- There are five things you need to do for file I/O
  1. Include the `fstream` header ⇔ iostream header
     - This gives access to the data types `ifstream` and `ofstream`
     - `Ifstream` ⇔ `istream`
     - `Ofstream` ⇔ `osteam`
  2. Declare file stream variables
     - Just like somewhere in the `iostream` header it declares:

```
ostream cout;   ⇔
ofstream outFileStream;
```

File Input/Output

- Five things for file I/O (continued)

3. Connect your new file stream variable to a file, and open it for reading (`ifstream`) or writing (`ofstream`)

```
outFileStream.open( “somefile.txt” )
outFileStream.open( “c:\\somefile.txt” )
```

4. Read from the file or write to the file
    - Same syntax as reading/writing to the cin/cout streams

```
outFileStream << “Put this in a file” << endl;
inFileStream >> x >> y >> z;
getline( inFileStream, myLine );
```

5. **Close the files when you’re done reading/writing**

```
outFileStream.close();
```

Overwrite vs. Append Modes

- Files may be opened with different modes
  - `open()` has an optional second argument to specify the mode
- By default, output file streams overwrite an existing file
- To append (add to the existing file):

```
outFile.open( "c:\\hw2Output.txt", ios::app );
//app ⇔ append
```

Example Case: End of File (EOF)

- Use a `while` loop to read from a file until you reach the end
  1. Initialization (before the loop)
     - Declare an `ifstream` variable
     - Open the file you want to read from
  2. Condition (the while condition)
     - Check to see if you've reached the end of the file
       – If it does, quit the loop
  3. Update (in the body of the loop)
     - Get new input *from the file stream*
     - Do something with that input
  4. Steps 2 and 3 repeat

Checking for End of File (EOF)

- Using the input stream as a condition, it is:
  - `false` if it is in an error state
  - `false` if it has *tried to read* an EOF
  - `true` otherwise
- You can also check explicitly by calling:

  `inputStreamVariable.eof()`

  - (returns true if the stream is at the end of the file)