std::getline, File I/O

ITP 165 – Fall 2015

Week 8, Lecture 1

Problem: Input and Multiple Words

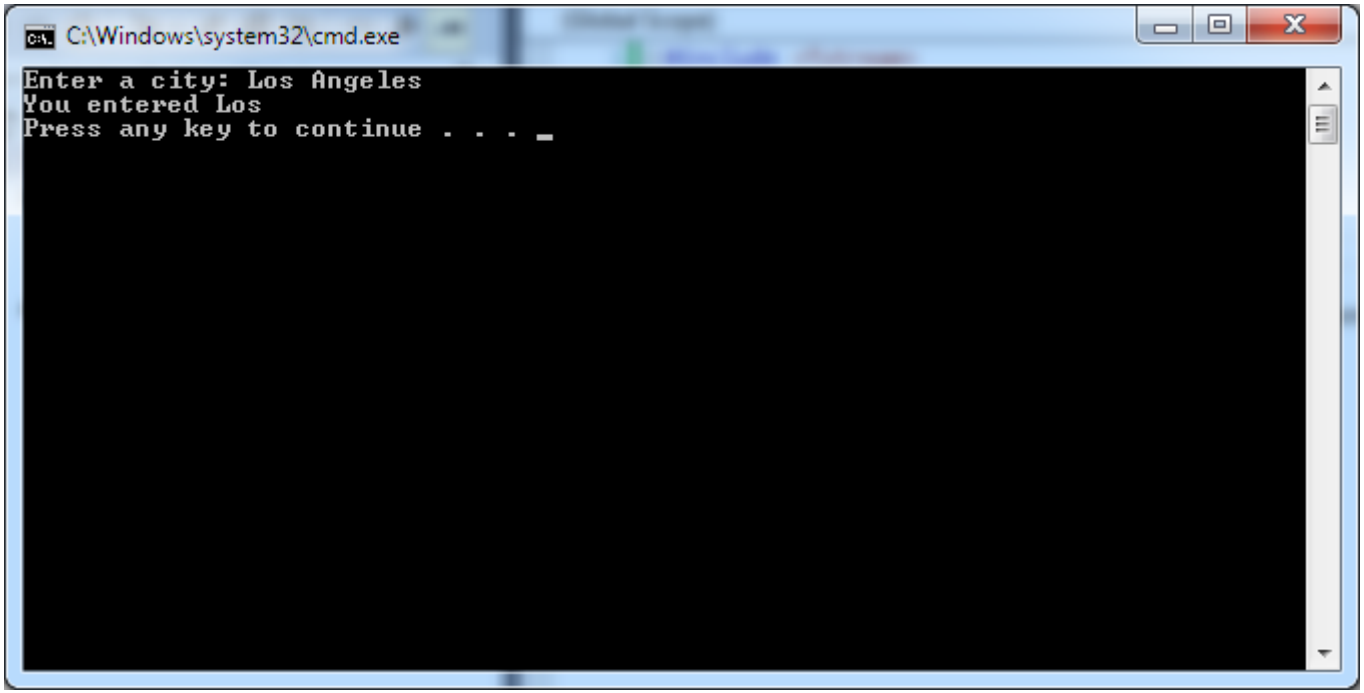
- Remember in Mad Libs, we couldn't accept multiple word phrases.

- For example, if I run this code:

```
std::cout << "Enter a city: ";
std::string city;
std::cin >> city;
std::cout << "You entered " << city << std::endl;
```

- If I type in "Los Angeles", what happens?

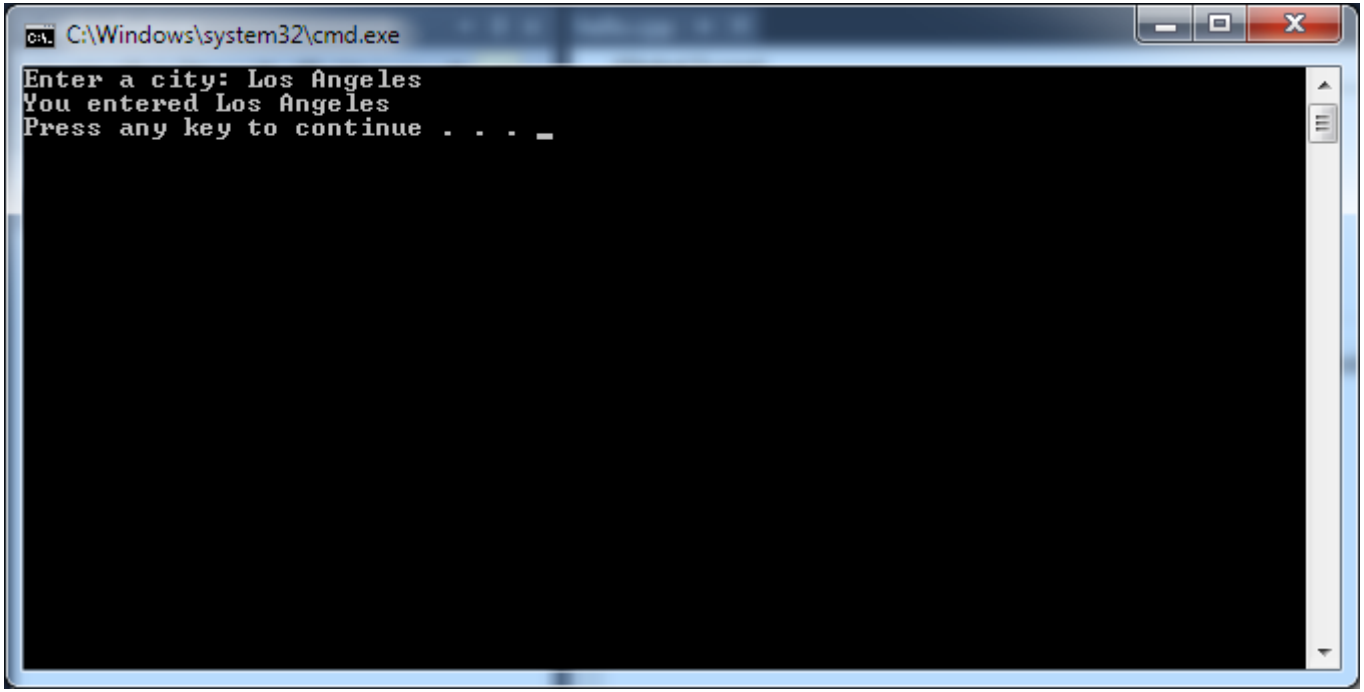
Problem: Input and Multiple Words



- It only accepted the first word, because by default `std::cin` stops at the first whitespace (space, tab, or enter)

Problem: Input and Multiple Words

- We want it instead to work like this!



Solution: std::getline

- In the `<string>` library, there is a function called `std::getline`

- `std::getline` allows you to get an entire line of input (so it doesn't stop until an enter)

- `std::getline` takes two parameters:
  - The input stream you want to grab the line from (what's an input stream?!)
  - The string that you want to store the line in

What is a stream?

- A ***stream*** is just another word for something that we will either get text input from, or write text output to
- So std::cout is an output stream that writes output to the console
- std::cin is an input stream that reads input from the console
- (As we'll see, there are potentially other input and output streams!)

std::getline in action

- So going to our previous example, we can use the std::getline function to grab the whole line:

```
std::cout << "Enter a city: ";
std::string city;
// std::getline takes two parameters:
//       - the input stream
//       - the string to store the line in
std::getline(std::cin, city);
std::cout << "You entered " << city << std::endl;
```

cin and getline not playing well

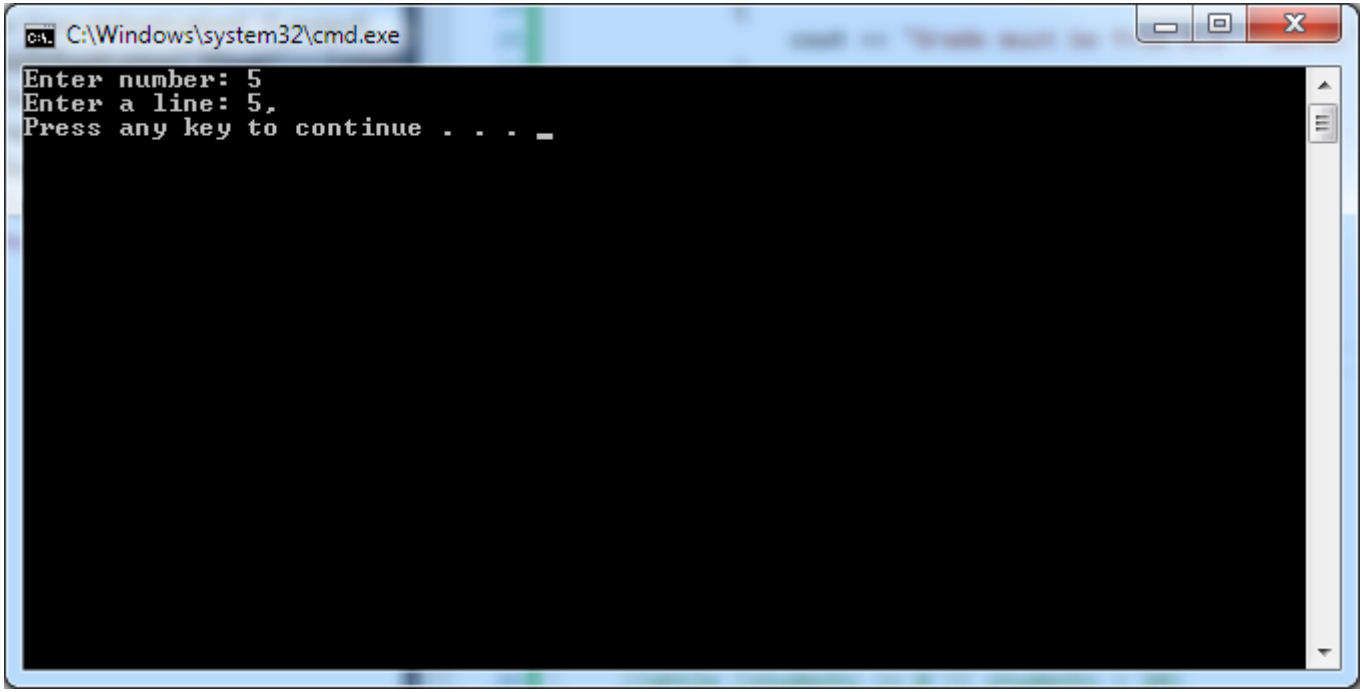
- If you do a std::cin and immediately follow it by a std::getline from cin, your std::getline will get ignored:

```
std::cout << "Enter number: ";
int num = 0;
std::cin >> num;

std::cout << "Enter a line: ";
std::string line;
std::getline(std::cin, line);

std::cout << num << "," << line << std::endl;
```

WHY
IGNORE
WHAT RETURN

Cin/getline not playing well

- It skips over the getline ☹

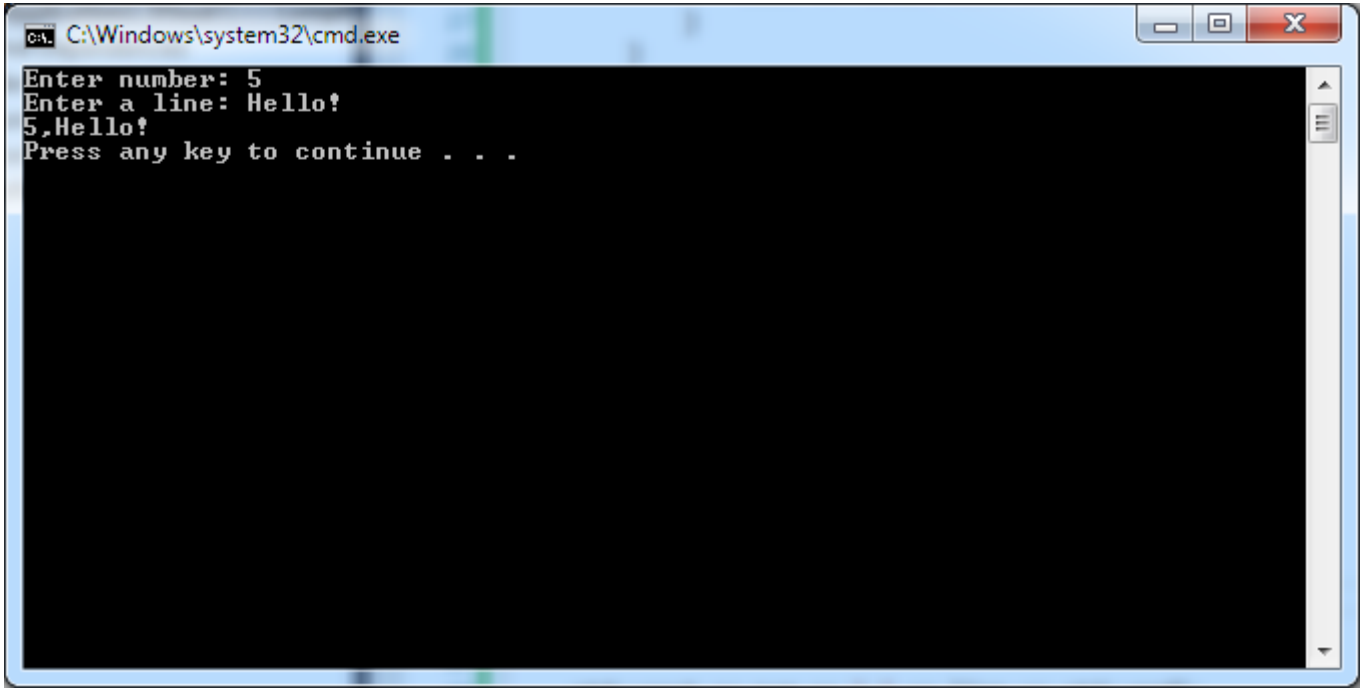


Cin/getline not playing well

- Add a std::cin.ignore() call:

```
std::cout << "Enter number: ";
int num = 0;
std::cin >> num;

std::cout << "Enter a line: ";
std::string line;

// NEED THIS IGNORE TO WORK
std::cin.ignore();

std::getline(std::cin, line);

std::cout << num << "," << line << std::endl;
```

Cin/getline problem fixed!



File Input/Output

- So far, we've only talked about Console Input/Output (I/O)…
- But what about a program like…

- It needs to read and write to files…we can do this with I/O streams that are specific to files

- So instead of cin/cout, we'll use different streams

A new library: fstream

- The `<fstream>` library handles streams for file input and output

- One big difference when using file streams is that unlike cin/cout, you have to declare a file stream before you use it

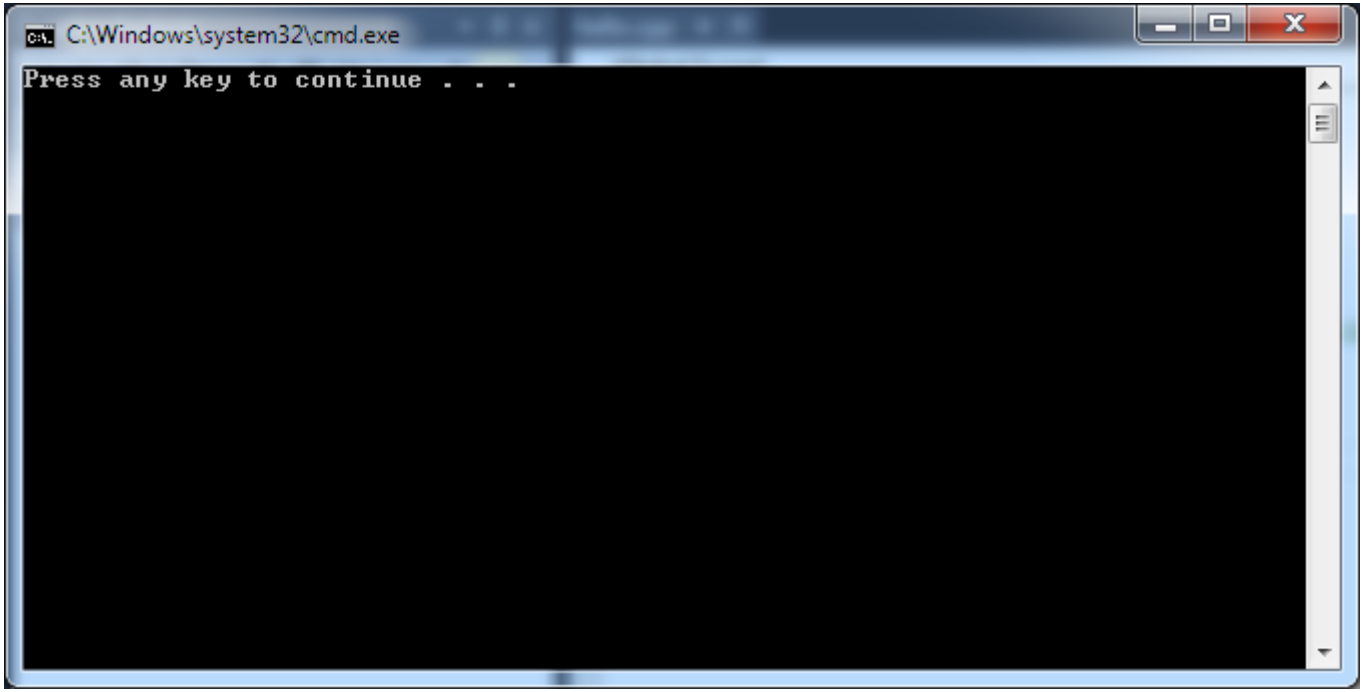
- This is because the file stream needs to know which file it's reading in or writing to!

Example: File output

```
#include <fstream>

int main()
{
    // Open the file for output
    std::ofstream fileStream("output.txt");

    // Write to the file output stream we declared
    fileStream << "Hello, World!";
    return 0;
}
```

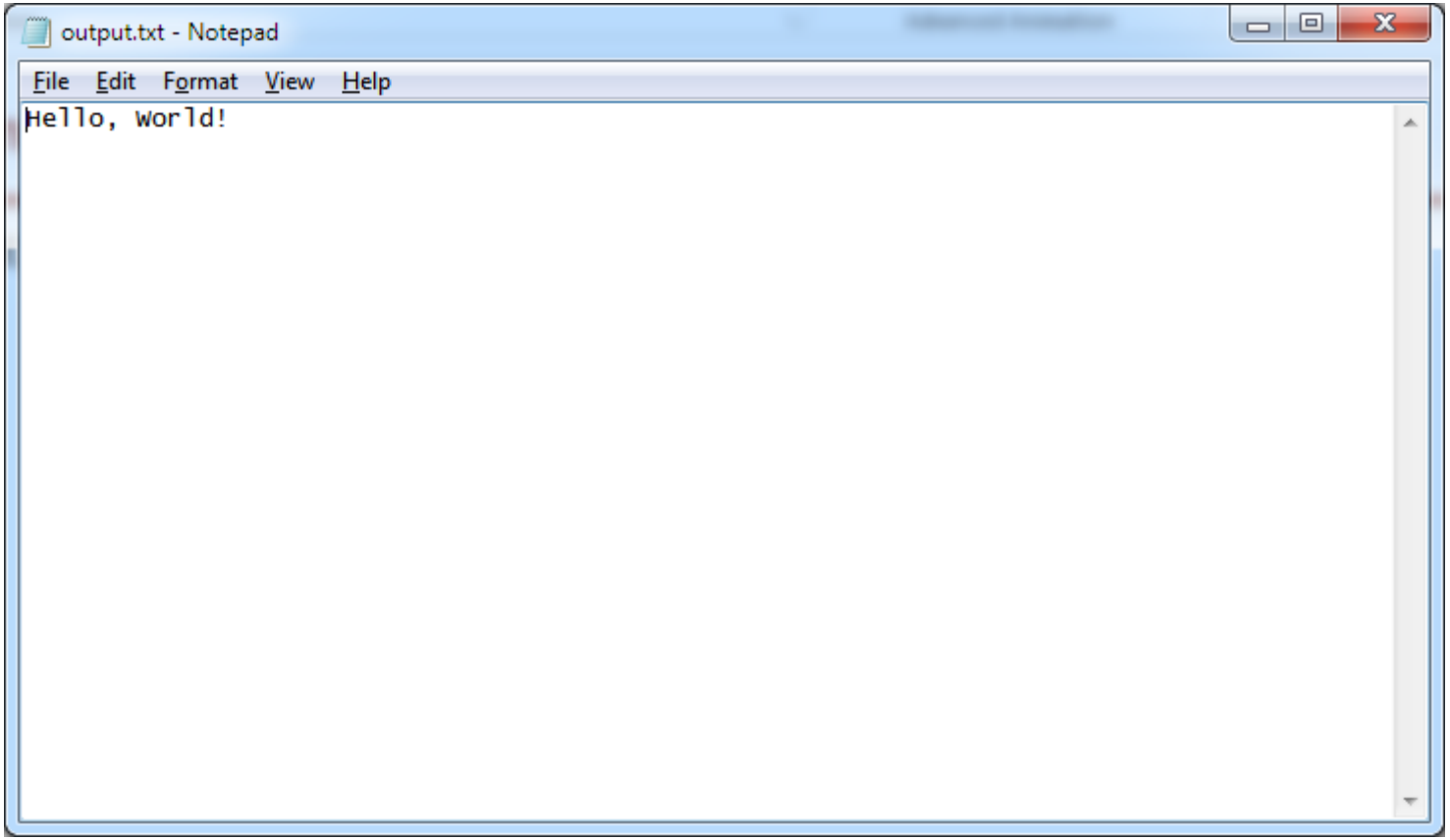
Example: File output

- If we run this program, we won't see any output to the console:



Example: File output

- But if we find the file named “output.txt”, it will have the output!



Declaring a File Output Stream

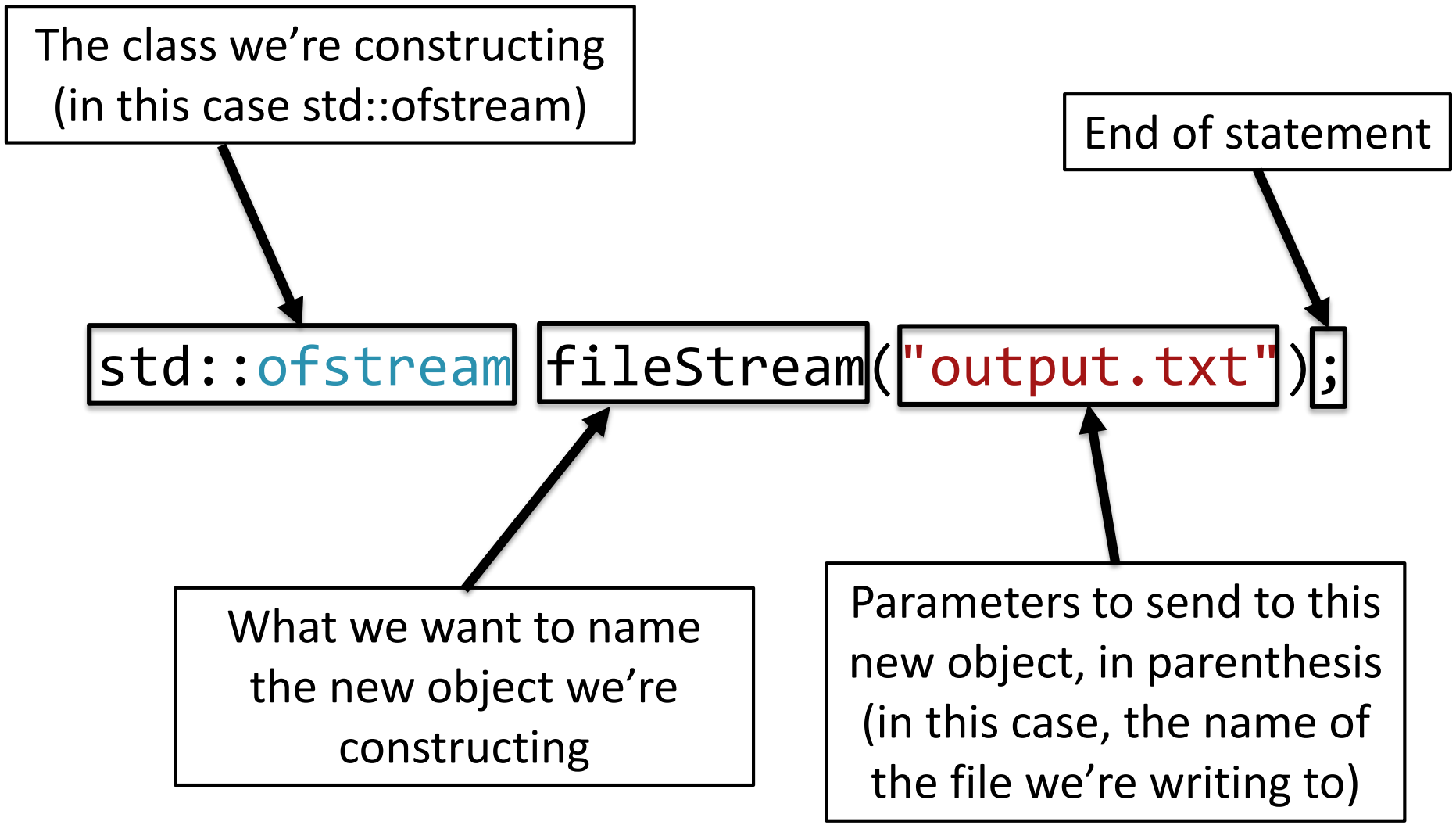
- The syntax for this looks a little weird:

```
std::ofstream fileStream("output.txt");
```

- "ofstream" stands for "output file stream"

- The reason why it looks like this – we are actually instantiating an instance of the `std::ofstream` class (also called ***constructing*** the object)!

Constructing an Object – Syntax



Writing to this new stream object

- To write output to this new file output stream object, it looks pretty much like a cout:

```
// Write to the file output stream we declared
fileStream << "Hello, World!";
```

Full Example, One More Time

```
#include <fstream>

int main()
{
    // Open the file for output
    std::ofstream fileStream("output.txt");

    // Write to the file output stream we declared
    fileStream << "Hello, World!";
    return 0;
}
```

File Output Caveat

- By default, if a file already exists by the name we specify, it will be overwritten!

- So in our case, if we already have a file named “output.txt”, we will lose all of its old contents

- There’s no warning, so be careful not to open a file you don’t want to lose!

- (So don’t do this):

```
std::ofstream oops("myLifesWork.cpp");
```

Constructing a File Input Stream

- It looks very similar to file output…

```
// Open "numbers.txt" for input
// Name this object fileInput
std::ifstream fileInput("numbers.txt");
```

- Only difference is we used an “ifstream” (input file stream) instead of an “ofstream” (output file stream)

Reading from a File Input Stream

- Suppose I have a file called numbers.txt that has two numbers in it:

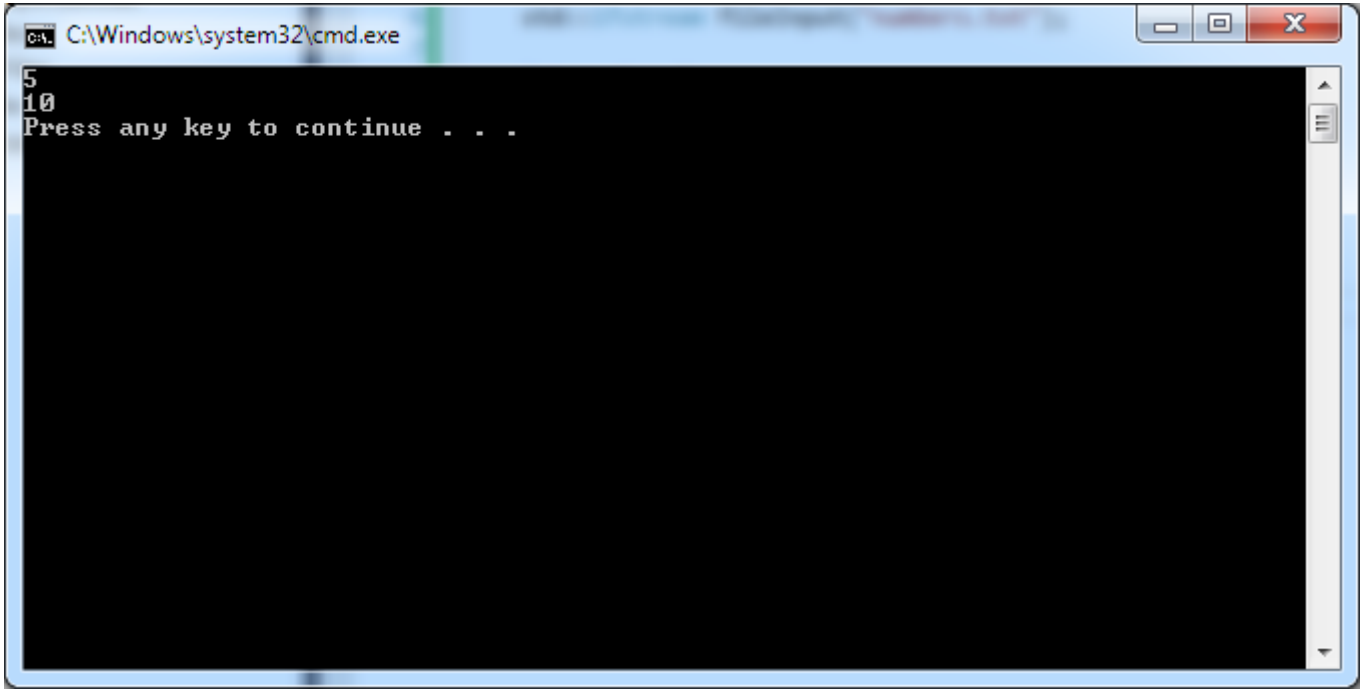
```
5
10
```

- Since they're numbers, I could use the input stream much like I'd use cin…

Reading from a File Input Stream - Example

```
#include <fstream>
#include <iostream>

int main() {
   // Open the file "numbers.txt" for input
   std::ifstream fileInput("numbers.txt");

   int num1 = 0;
   int num2 = 0;

   fileInput >> num1;
   fileInput >> num2;

   std::cout << num1 << std::endl;
   std::cout << num2 << std::endl;

   return 0;
}
```

Reading from a File Input Stream - Example



- So this works, but there are major problems with the code!

Reading from a File Input Stream - Caveats

- To make sure our code works properly when using file input, there are two issues that we have to confront:

1. What if the file we try to open doesn't exist?

2. How do we know when we've read in the whole file?

The is_open member function

- `std::ifstream` has a member function called `is_open` that tells us whether or not the file is open

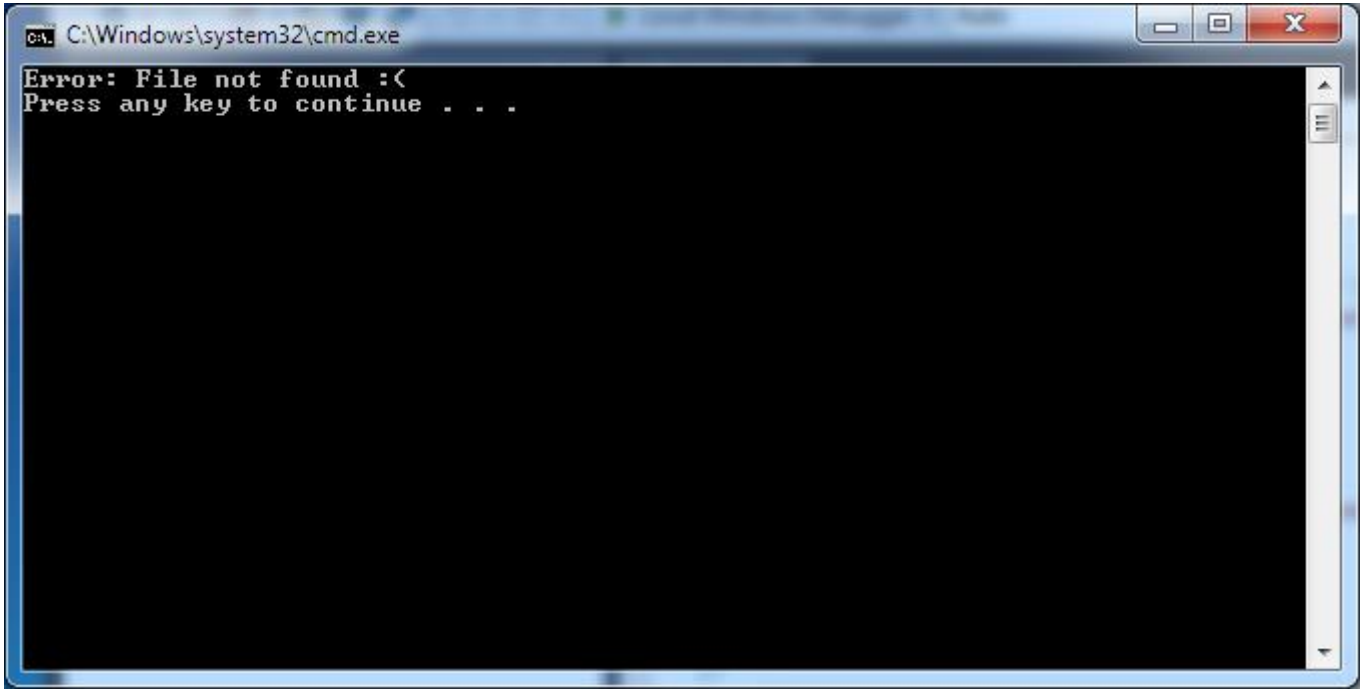
- `is_open` –
  - Takes no parameters
  - Returns a `bool` – true if the file is open, false if not

is_open in action

```
#include <fstream>
#include <iostream>

int main() {
   // Open the file "randomfile.txt" for input
   std::ifstream fileInput("randomfile.txt");

   if (fileInput.is_open()) {
      // TODO: Do stuff with the file
   }
   else {
      std::cout << "Error: File not found :(" << std::endl;
   }

   return 0;
}
```

is_open in action

- So if randomfile.txt doesn't exist:



The eof member function

- The eof member function tells us when we've reached the EOF (End of File)

- `eof` –
  - Takes no parameters
  - Returns a `bool` – true if we're at the end of the file, false if not

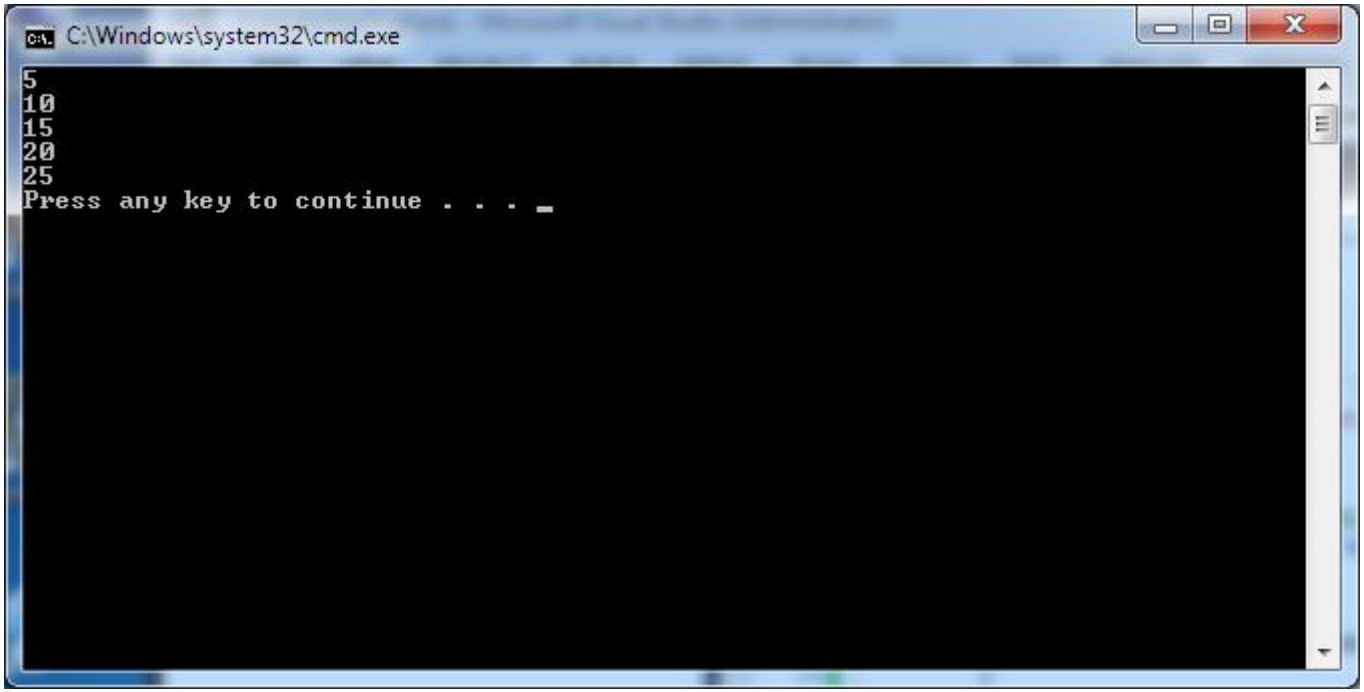
Numbers, Revisited

```
#include <fstream>
#include <iostream>

int main() {
   // Open the file "numbers.txt" for input
   std::ifstream fileInput("numbers.txt");

   if (fileInput.is_open()) {
      // Continue looping as long as not at EOF!
      while (fileInput.eof() != true) {
         int num;
         fileInput >> num;
         std::cout << num << std::endl;
      }
   }
   else {
      std::cout << "Error: File not found :(" << std::endl;
   }

   return 0;
}
```

Numbers, Revisited

- So if numbers.txt has:

```
5
10
15
20
25
```



File Input and std::getline

- Just like with std::cin, I can use std::getline on file input streams

- For example, suppose I have a file called `story.txt` with the following:

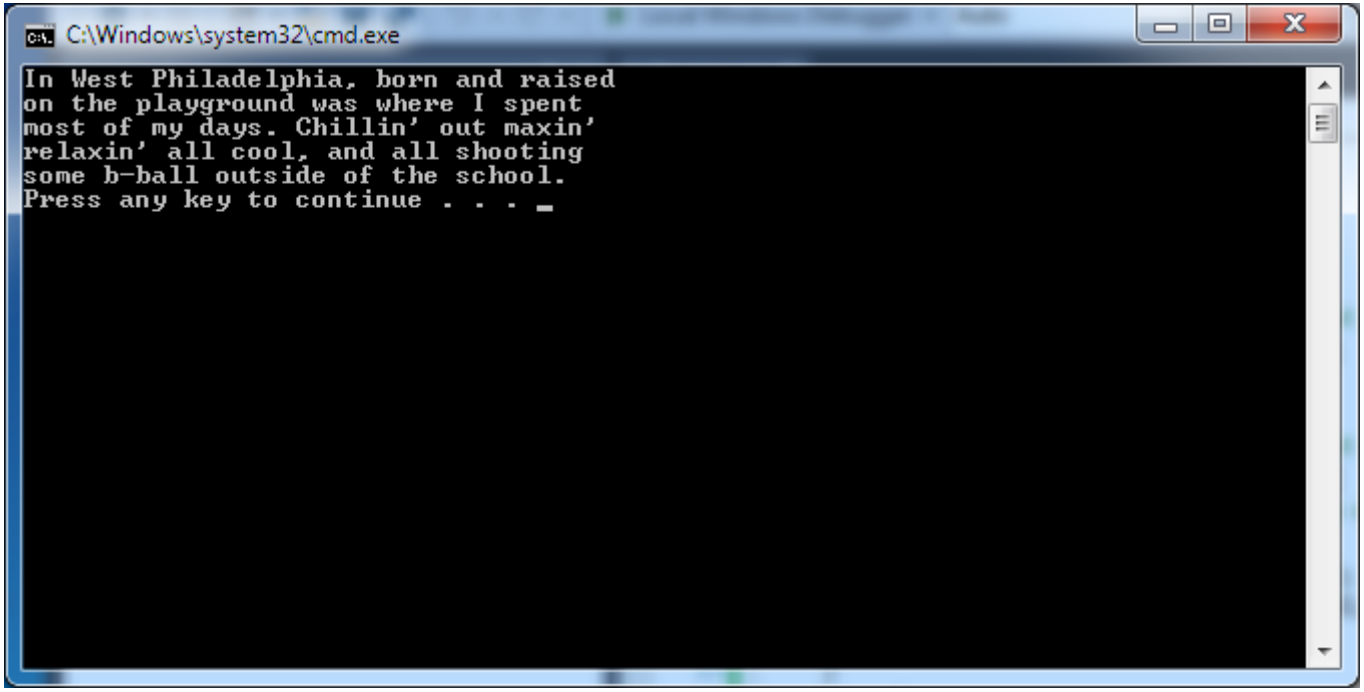
```
In West Philadelphia, born and raised
on the playground was where I spent
most of my days. Chillin' out maxin'
relaxin' all cool, and all shooting
some b-ball outside of the school.
```

- I could write a program that reads in each line, and writes it back to cout…

File Input and std::getline in action

```
#include <fstream>
#include <iostream>
#include <string>

int main() {
   // Open the file "story.txt" for input
   std::ifstream fileInput("story.txt");

   if (fileInput.is_open()){
      // Continue looping as long as not at EOF!
      while (fileInput.eof() != true) {
         // Use std::getline to grab a whole line
         std::string line;
         std::getline(fileInput, line);
         std::cout << line << std::endl;
      }
   }
   else {
      std::cout << "Error: File not found :(" << std::endl;
   }

   return 0;
}
```

File Input and std::getline in action



Lab Practical #13