EECS 183, Week 8

Stream States

A stream, such as `cin`, is always in one of four states.

State	Meaning
good	
bad	A fatal error occurred (e.g. the program tried to read past the end of a file)
eof	
fail	A non-fatal error occurred (e.g. the user entered letters when the program was expecting an integer)

Each state has a corresponding member function to check if a stream is (not) in that state. `clear()` member function will make the state good gain:

```
if (!cin.good()) {
    cin.clear();
}
```

How would you go to the next line of input if input is invalid?

positive.cpp

Implement `getPositiveInt`, a function that keeps asking the user for a positive integer until the user cooperates. This function should handle all bad input.

```
int getPositiveInt() {




}

int main() {
    cout << "Please give me a positive integer: ";
    int n = getPositiveInt();
    cout << "Thanks for the " << n << "!" << endl;
}
```

File I/O

Below is a sample program that opens a file called `input.txt`, reads a single word outputs the word to `output.txt`.

```
#include <fstream>
...
ifstream inputFileStream;
inputFileStream.open("input.txt");
if (inputFileStream.good()) {
    string name;
    inputFileStream >> name;
    ofstream outputFileStream;
    outputFileStream.open("output.txt");
    if (outputFileStream) {
        outputFileStream << name;
    }
}
```

More on files

1. Write a program that asks user for a filename and prints the contents of that file.

2. Write a program that asks user for two filenames and copies the contents (each line) of the first file into the second file (new or overwritten).

3. Write a program that asks user for two filenames and copies each line of the first file into the second file (new or overwritten), in **reverse order**.

Classes

Classes allow us to encapsulate _____ and _____.

They are also a way to create new types in C++. Here's a class that lets us represent colors:

```
class Color {
public:
    int red;
    int green;
    int blue;
};
```

To use the `Color` class, create variables of type `Color` and use dot (`.`) notation to access its members:

```
Color whiteColor;
whiteColor.red = 255;
whiteColor.green = 255;
whiteColor.blue = 255;
```

However, we tend to make data (i.e., member variables) private, so it is not directly accessible. Instead, we access it via **getter** and **setter** functions.

Each class is defined in its own header (`.h`) file. Member (instance) functions are implemented in the corresponding `.cpp` file. To create objects of a class (in `main.cpp`), include the `.h` file.

Color.h	Color.cpp	main.cpp

<pre>class Color {
private:
    int red;
    int green;
    int blue;
public:
    int getRed();
    void setRed(int r);
    int getGreen();
    void setGreen(int g);
    int getBlue();
    void setBlue(int b);
};</pre>
	<pre>#include "Color.h"
	
	int Color::getRed() {
	    return red;
	}
	
	int Color::getGreen() {
	    return green;
	}
	
	int Color::getBlue() {
	    return blue;
	}
	
	// ...</pre>
		<pre>#include <iostream>
		#include "Color.h"
		using namespace std;
		
		Student getStudent();
		
		int main() {
		    Color whiteColor;
		    whiteColor.setRed(255);
		    whiteColor.setGreen(255);
		    whiteColor.setBlue(255);
		
		    // ...
		}</pre>