coursera

This reading explains a few details to keep in mind about how C++ programs are structured in code. Because a C++ program's source code exists across several separate files, you need to know how the compiler will pull those pieces together.

If you look at the example source code provided so far, you'll see that C++ code files often come with two different file extensions:

.h files are "header files". These usually have definitions of objects and declarations of global functions. Recently, some people name header files with a ".hpp" suffix instead.

.cpp files are often called the "implementation files," or simply the "source files". This is where most function definitions and main program logic go.

In general, the header files contain *declarations* (where classes and custom types are listed by name and type, and function prototypes give functions a type signature) while the source files contain *implementation* (where function bodies are actually defined, and some constant values are initialized). It becomes easier to understand this organizational separation if you know more about how the code is compiled into a program you can run. Let's look at the **cpp-class** example from the provided code.

The Cube.h header file has this content:

```
1   /**
2    * Simple C++ class for representing a Cube.
3    *
4    * @author
5    *    Wade Fagen-Ulmschneider <waf@illinois.edu>
6    */
7
8   // All header (.h) files start with "#pragma once":
9   #pragma once
10
11  // A class is defined with the `class` keyword, the name
12  // of the class, curly braces, and a required semicolon
13  // at the end:
14  class Cube {
15    public:  // Public members:
16      double getVolume();
17      double getSurfaceArea();
18      void setLength(double length);
19
20    private: // Private members:
21      double length_;
22  };
23
```

Note the **#pragma once** at the beginning. Instructions beginning with **#** are special commands for the compiler, called preprocessor directives. This instruction prevents the