Copy Constructor
When a non-primitive variable is passed/returned by value, a copy must be made. As with a constructor, an automatic copy constructor is provided for you if you choose not to define one:

All copy constructors will:

The automatic copy constructor:
1. 
2. 

To define a custom copy constructor:

```cpp
class Cube {
public:
    Cube(); // default ctor
    Cube(double length); // 1-param ctor
    double getVolume();
    double getSurfaceArea();
private:
    double length_; 
};
```

Recall the joinCubes function:

```cpp
Cube joinCubes(Cube c1, Cube c2) {
    double totalVolume = c1.getVolume() + c2.getVolume();
    double newLength = std::pow( totalVolume, 1.0/3.0 );
    Cube result(newLength); 
    return result; 
}
```

Bringing Concepts Together:
How many times do our different joinCubes files call each constructor?

<table>
<thead>
<tr>
<th></th>
<th>By Value</th>
<th>By Pointer</th>
<th>By Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cube(double)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cube(const Cube&amp;)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cubes Unite!
Consider a Tower made of three Cubes:

```cpp
#pragma once
#include "cs225/Cube.h"
using cs225::Cube;

class Tower {
public:
    Tower(Cube c, Cube *ptr, const Cube &ref);
    Tower(const Tower & other);
private:
    Cube cube_; 
    Cube *ptr_; 
    const Cube &ref_; 
};
```

Automatic Copy Constructor Behavior:
The behavior of the automatic copy constructor is to make a copy of every variable. We can mimic this behavior in our Tower class:

```cpp
# Tower.cpp
Tower::Tower(const Tower & other) 
    : cube_(other.cube_), 
      ptr_(other.ptr_), 
      ref_(other.ref_) 
{ }
```

...we refer to this as a ______________________ because:
**Deep Copy via Custom Copy Constructor:**
Alternatively, a custom copy constructor can perform a deep copy:

```cpp
tower.cpp
11  Tower::Tower(const Tower & other) {
12    // Deep copy cube_
13    // Deep copy ptr_
14    // Deep copy ref_
15  }
```

**Destructor**
The last and final member function called in the lifecycle of a class is the destructor.

**Purpose of a destructor:**

The **automatic destructor:**

1. 
2. 

**Custom Destructor:**

---

**Overloading Operators**
C++ allows custom behaviors to be defined on over 20 operators:

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>+</th>
<th>-</th>
<th>*</th>
<th>/</th>
<th>%</th>
<th>++</th>
<th>--</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitwise</td>
<td>&amp;</td>
<td>^</td>
<td>~</td>
<td>&lt;&lt;</td>
<td>&gt;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assignment</td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comparison</td>
<td>==</td>
<td>!=</td>
<td>&gt;</td>
<td>&lt;</td>
<td>&gt;=</td>
<td>&lt;=</td>
<td></td>
</tr>
<tr>
<td>Logical</td>
<td>!</td>
<td>&amp; &amp;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>[ ]</td>
<td>()</td>
<td>- &gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**General Syntax:**

**Adding overloaded operators to Cube:**

---

**CS 225 – Things To Be Doing:**

1. Theory Exam #1 starts **this Thursday**, covers through today
2. MP1 due tonight; grace period until Tuesday @ 11:59pm
3. MP2 released on Tuesday *(start early for extra credit!!)*
4. Lab Extra Credit → Attendance in your registered lab section!
5. Daily POTDs every M-F for daily extra credit!