## Returning from a function

Identical to passing into a function, we also have three choices on how memory is used when returning from a function:

Return by value:

```cpp
Cube joinCubes(const Cube &s1, const Cube &s2)
```

Return by reference:

```cpp
Cube &joinCubes(const Cube &s1, const Cube &s2)
```

...remember: never return a reference to stack memory!

Return by pointer:

```cpp
Cube *joinCubes(const Cube &s1, const Cube &s2)
```

...remember: never return a reference to stack memory!

## Copy Constructor

When a non-primitive variable is passed/returned by value, a copy must be made.

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_;
};
```

## 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_;  // deep copy
    Cube *ptr_;  // deep copy
    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::Tower(const Tower & other) {
  cube_ = other.cube_;  // deep copy
  ptr_ = other.ptr_;    // deep copy
  ref_ = other.ref_;    // deep copy
}
```

...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::Tower(const Tower & other) {  // Deep copy cube_
  cube_ = other.cube_;  // deep copy
  ptr_ = other.ptr_;    // deep copy
  ref_ = other.ref_;    // deep copy
}
```
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. Like a constructor and copy constructor, an automatic destructor exists only when no custom destructor is defined.
2. [Invoked]:
3. [Functionality]:

Custom Destructor:

<table>
<thead>
<tr>
<th>Cube.h</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
</tbody>
</table>

...necessary if you need to delete any heap memory!

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

| Arithmetic | +  -  *  /  %  ++  -- |
| Bitwise    | &  |  ~  <  >  |
| Assignment | =  |
| Comparison | ==  ! =  >  <  >=  <= |
| Logical    | !  &&  || |
| Other      | []  ()  -> |

General Syntax:
Adding overloaded operators to Cube:

<table>
<thead>
<tr>
<th>Cube.h</th>
<th>Cube.cpp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>/* ... */</td>
</tr>
<tr>
<td>3</td>
<td>41</td>
</tr>
<tr>
<td>4</td>
<td>42</td>
</tr>
<tr>
<td>10</td>
<td>43</td>
</tr>
<tr>
<td>11</td>
<td>44</td>
</tr>
<tr>
<td>12</td>
<td>45</td>
</tr>
<tr>
<td>13</td>
<td>46</td>
</tr>
<tr>
<td>14</td>
<td>47</td>
</tr>
</tbody>
</table>

One Very Powerful Operator: Assignment Operator

<table>
<thead>
<tr>
<th>Cube.h</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cube.cpp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Functionality Table:

<table>
<thead>
<tr>
<th>Copies an object</th>
<th>Destroys an object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy constructor</td>
<td></td>
</tr>
<tr>
<td>Copy Assignment</td>
<td></td>
</tr>
<tr>
<td>Destructor</td>
<td></td>
</tr>
</tbody>
</table>

The Rule of Three
If it is necessary to define any one of these three functions in a class, it will be necessary to define all three of these functions:
1. 
2. 
3. 
The Rule of Zero

CS 225 and Rule Three/Five/Zero
In CS 225 We will:

CS 225 – Things To Be Doing:
1. Finish lab_intro
2. Start on mp_stickers
3. First PotD released today due Monday.