Contrasting the three methods:

<table>
<thead>
<tr>
<th>By Value</th>
<th>By Pointer</th>
<th>By Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exactly what is copied when the function is invoked?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Does modification of the passed in object modify the caller’s object?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is there always a valid object passed in to the function?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Speed

Safety

Using the \texttt{const} keyword

1. Using \texttt{const} in function parameters:

2. Using \texttt{const} as part of a member functions’ declaration:

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:

Return by reference:

Return by pointer:

Copy Constructor

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

All \texttt{copy constructors} will:

The \textbf{automatic copy constructor}:

1.  

2.  

To define a \textbf{custom copy constructor}:

```cpp
class Cube {
  public:
    Cube(); // default ctor
    Cube(double length); // 1-param ctor
    double getVolume();
    double getSurfaceArea();
  private:
    double length_; 
};
```
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_;  // Deep copy
    Cube *ptr_;  // Deep copy
    const Cube &ref_;  // Deep copy
};
```

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
}
```

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_
    // Deep copy ptr_
    // Deep copy ref_
}
```

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

Purpose of a destructor:

The automatic destructor:

```cpp
class Cube {
public:
    Cube();  // default ctor
    Cube(double length);  // 1-param ctor
    Cube(const Cube & other);  // custom copy ctor
    ~Cube();  // destructor, or dtor
};
```

CS 225 – Things To Be Doing:

1. lab_intro and lab_debug due Sunday @ 11:59pm
2. Mp_intro is due Tuesday @ 11:59pm
3. Daily POTDs every weekday for daily extra credit!