



# CS 225

## Data Structures

*September 9 – Overloading*

*G Carl Evans*



# Destructor

**[Purpose]:**



# Destructor

**[Purpose]:** Free any resources maintained by the class.

## **Automatic Destructor:**

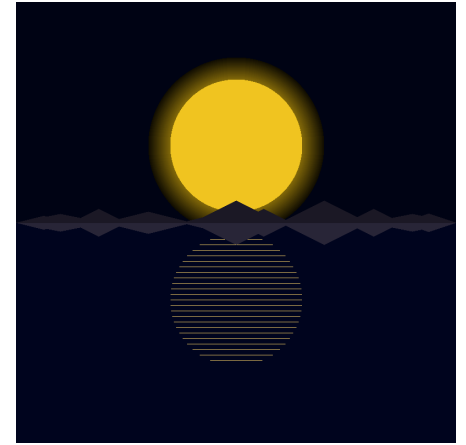
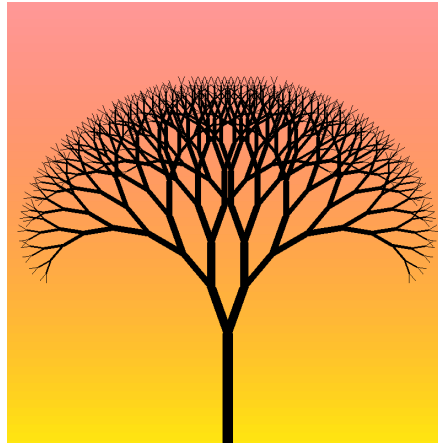
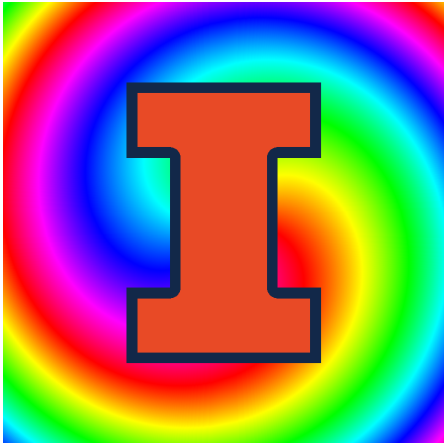
1. Exists only when no custom destructor is defined.
2. [Invoked]:
3. [Functionality]:

## cs225/Cube.h

```
1 #pragma once
2
3 namespace cs225 {
4     class Cube {
5     public:
6         Cube();
7         Cube(double length);
8         Cube(const Cube & other);
9         ~Cube();
10
11         double getVolume() const;
12         double getSurfaceArea() const;
13
14     private:
15         double length_;
16     };
17 }
18
19
20
```

## cs225/Cube.cpp

```
7 namespace cs225 {
8     Cube::Cube() {
9         length_ = 1;
10        cout << "Default ctor"
11            << endl;
12    }
13
14    Cube::Cube(double length) {
15        length_ = length;
16        cout << "1-arg ctor"
17            << endl;
18    }
19
20
21
22
23
24
25
... // ...
```



MP 1 Art

---

## Operators that can be overloaded in C++

<b>Arithmetic</b>	<code>+</code>	<code>-</code>	<code>*</code>	<code>/</code>	<code>%</code>	<code>++</code>	<code>--</code>
<b>Bitwise</b>	<code>&amp;</code>	<code> </code>	<code>^</code>	<code>~</code>	<code>&lt;&lt;</code>	<code>&gt;&gt;</code>	
<b>Assignment</b>	<code>=</code>						
<b>Comparison</b>	<code>==</code>	<code>!=</code>	<code>&gt;</code>	<code>&lt;</code>	<code>&gt;=</code>	<code>&lt;=</code>	
<b>Logical</b>	<code>!</code>	<code>&amp;&amp;</code>	<code>  </code>				
<b>Other</b>	<code>[]</code>	<code>()</code>	<code>-&gt;</code>				

## cs225/Cube.h

```
1 #pragma once
2
3 namespace cs225 {
4     class Cube {
5     public:
6         Cube();
7         Cube(double length);
8         Cube(const Cube & other);
9         ~Cube();
10
11
12
13
14
15         double getVolume() const;
16         double getSurfaceArea() const;
17
18     private:
19         double length_;
20     };
}
```

## cs225/Cube.cpp

```
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
```



# One Very Special Operator

**Definition Syntax (.h):**

`Cube & operator=(const Cube& s)`

**Implementation Syntax (.cpp):**

`Cube & Cube::operator=(const Cube& s)`





# Assignment Operator

**Similar to Copy Constructor:**

**Different from Copy Constructor:**

# Assignment Operator

	Copies an object	Destroys an object
Copy constructor		
Copy Assignment operator		
Destructor		



## MP: Extra Credit

**The most successful MP is an MP done early!**

Unless otherwise specified in the MP, we will award up to 8 points of extra credit for completing part 1 by the extra credit deadline (the Monday following the release of the MP)

**Scaled by tests passed**

Example on MP 2 (19 tests)

19/19 = 8 points EC

18/19 = 7.58 points EC

17/19 = 7.16 points EC

16/19 = 6.74 points EC

...



## 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”

## Corollary to Rule of Five

Classes that **declare** custom destructors, copy/move constructors or copy/move assignment operators should deal exclusively with ownership. Other classes should not **declare** custom destructors, copy/move constructors or copy/move assignment operators

–Scott Meyers



In CS 225



# Rvalue Reference or Move Semantics

- Rvalue

- Move

```
Cube (const Cube&& s) noexcept
```

- Move Assignment

```
Cube & operator=(const Cube&& s) noexcept
```



## The “Rule of Five”

If it is necessary to define any one of these five functions in a class, it will be necessary to define all five of these functions:

1.

2.

3.

4.

5.