



# CS 225

## Data Structures

*September 11 - Inheritance*

*G Carl Evans*



# Destructor

**[Purpose]:**



# Destructor

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

## **Automatic Destructor:**

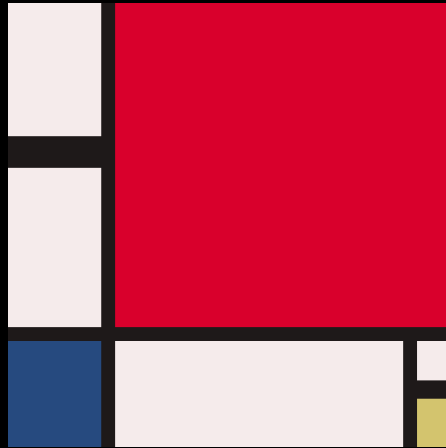
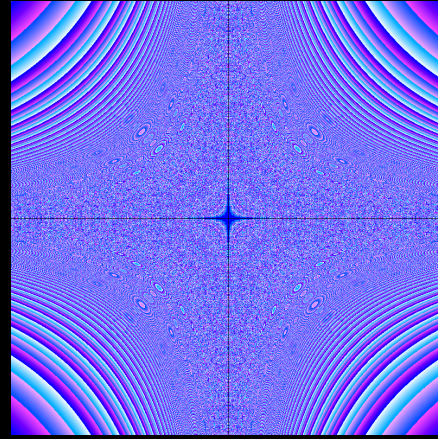
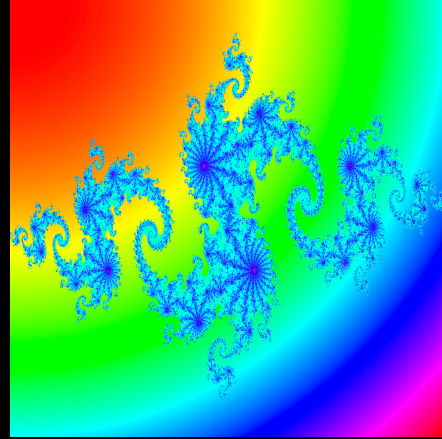
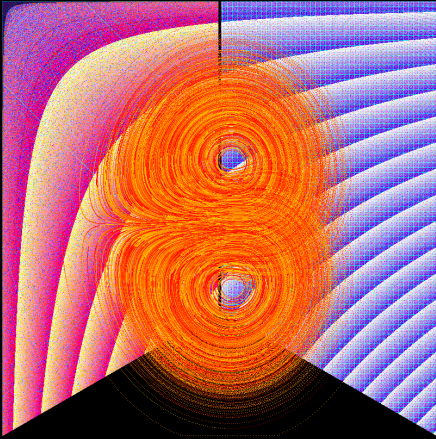
1. Exists only when no custom destructor is defined.
2. [Invoked]:
3. [Functionality]:

## 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
<b>Copy constructor</b>		
<b>Assignment operator</b>		
<b>Destructor</b>		



## MP: Extra Credit

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

Unless otherwise specified in the MP, we will award +1 extra credit point per day **for completing Part 1** before the due date (*up to +7 points*):

### Example for MP2:

**+7** points: Complete by **Monday**, Sept. 16 (11:59pm)

**+6** points: Complete by **Tuesday**, Sept. 17 (11:59pm)

**+5** points: Complete by **Wednesday**, Sept. 18 (11:59pm)

**+4** points: Complete by **Thursday**, Sept. 19 (11:59pm)

**+3** points: Complete by **Friday**, Sept. 20 (11:59pm)

**+2** points: Complete by **Saturday**, Sept. 21 (11:59pm)

**+1** points: Complete by **Sunday**, Sept. 22 (11:59pm)

**MP2 Due Date: Monday, Sept. 23**



## MP: Extra Credit

We will give **partial credit** and **maximize the value** of your extra credit:

You made a submission and missed a few edge cases in Part 1:

Monday:  $+7 * 80\% = +5.6$  earned



## MP: Extra Credit

We will give **partial credit** and **maximize the value** of your extra credit:

You made a submission and missed a few edge cases in Part 1:

Monday:  $+7 * 80\% = +5.6$  earned

You fixed your code and got a perfect score on Part 1:

Tuesday:  $+6 * 100\% = +6$  earned (*maximum benefit*)

## MP: Extra Credit

We will give **partial credit** and **maximize the value** of your extra credit:

You made a submission and missed a few edge cases in Part 1:

Monday:  $+7 * 80\% = +5.6$  earned

You fixed your code and got a perfect score on Part 1:

Tuesday:  $+6 * 100\% = +6$  earned (*maximum benefit*)

You began working on Part 2, but added a compile error:

Wednesday:  $+5 * 0\% = +0$  earned (*okay to score lower later*)

...



## 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.**



# Inheritance



## Shape.h

```
4 class Shape {
5     public:
6         Shape();
7         Shape(double length);
8         double getLength() const;
9
10    private:
11        double length_;
12 };
13
14
15
16
17
18
19
20
```

## Shape.cpp

```
8 Shape::Shape() {
9     length_ = 1;
10 }
11
12 Shape::Shape(double length) {
13     length_ = length;
14 }
15
16 double Shape::getLength()
17 const {
18     return length_;
19 }
20
21
22
23
24
25
26
27
28
...
```

## Square.h

```
1 #pragma once
2
3 #include "Shape.h"
4
5 class Square      {
6     public:
7         double getArea() const;
8
9     private:
10        // Nothing!
11 };
12
13
14
15
16
17
18
19
20
```

## Square.cpp

```
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
...
```

# Derived Classes

## [Public Members of the Base Class]:

main.cpp

```
5 int main() {  
6     Square sq;  
7     sq.getLength(); // Returns 1, the length init'd  
8                     // by Shape's default ctor  
...     ...  
... }
```

## [Private Members of the Base Class]:

## Square.h

```
1 #pragma once
2
3 #include "Shape.h"
4
5 class Square      {
6     public:
7
8         double getArea() const;
9
10    private:
11        // Nothing!
12 };
13
14
15
16
17
18
19
20
```

## Square.cpp

```
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
...
```

## Cube.h

```
4 class Cube {
5     public:
6         double getVolume() const;
7         double getSurfaceArea() const;
8
9     private:
10        // Nothing!
11 };
12
13
14
15
16
17
18
19
20
```

## Cube.cpp

```
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
...
```

## RubikCube.h

```
1 #pragma once
2
3 class RubikCube : public Cube {
4     public:
5         void solve();
6
7         void turnRow(int r);
8         void turnColumn(int c);
9         void rotate(int direction);
10
11     private:
12         // ...
13 };
14
15
16
17
18
19
20
21
22
```

## RubikCube.cpp

```
1 #include "RubikCube.h"
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```



Virtual

## Cube.cpp

```
1 Cube::print_1() {
2     cout << "Cube" << endl;
3 }
4
5 Cube::print_2() {
6     cout << "Cube" << endl;
7 }
8
9 virtual Cube::print_3() {
10     cout << "Cube" << endl;
11 }
12
13 virtual Cube::print_4() {
14     cout << "Cube" << endl;
15 }
16
17 // In .h file:
18 virtual Cube::print_5() = 0;
19
20
21
22
```

## RubikCube.cpp

```
1 // No print_1() in RubikCube.cpp
2
3
4
5 RubikCube::print_2() {
6     cout << "Rubik" << endl;
7 }
8
9 // No print_3() in RubikCube.cpp
10
11
12
13 RubikCube::print_4() {
14     cout << "Rubik" << endl;
15 }
16
17 RubikCube::print_5() {
18     cout << "Rubik" << endl;
19 }
20
21
22
```



# Runtime of Virtual Functions

<u>virtual-main.cpp</u>	Cube c;	RubikCube c;	RubikCube rc; Cube &c = rc;
c.print_1();			
c.print_2();			
c.print_3();			
c.print_4();			
c.print_5();			