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:

```cpp
#pragma once
class Cube {
public:
    Cube(); // default ctor
    Cube(double length); // 1-param ctor
    Cube(const Cube & other); // custom copy ctor
    ~Cube(); // destructor, or dtor
    ...
...necessary if you need to delete any heap memory!
```

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>~</td>
<td>&lt;&lt;</td>
<td>&gt;&gt;</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;</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:

<table>
<thead>
<tr>
<th>Cube.h</th>
<th>Cube.cpp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 #pragma once</td>
<td>... /* ... */</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3 class Cube {</td>
<td></td>
</tr>
<tr>
<td>4     public:</td>
<td></td>
</tr>
<tr>
<td>5     // ...</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7     // default ctor</td>
<td></td>
</tr>
<tr>
<td>8     // 1-param ctor</td>
<td></td>
</tr>
<tr>
<td>9     // custom copy ctor</td>
<td></td>
</tr>
<tr>
<td>10    // destructor, or dtor</td>
<td></td>
</tr>
<tr>
<td>... // ...</td>
<td>... /* ... */</td>
</tr>
</tbody>
</table>

One Very Powerful Operator: Assignment Operator

```cpp
Cube & operator=(const Cube & other);
```

<table>
<thead>
<tr>
<th>Cube.h</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube &amp; operator=(const Cube &amp; other);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cube.cpp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube &amp; Cube::operator=(const Cube &amp; other) { ... }</td>
</tr>
</tbody>
</table>

Functionality Table:

<table>
<thead>
<tr>
<th></th>
<th>Copies an object</th>
<th>Destroys an object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy constructor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assignment operator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destructor</td>
<td></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.
Inheritance
In nearly all object-oriented languages (including C++), classes can be extended to build other classes. We call the class being extended the **base class** and the class inheriting the functionality the **derived class**.

**Base Class: Shape**

```cpp
// Shape.h
4 class Shape {
5 public:
6    Shape();
7    Shape(double length);
8    double getLength() const;
9 private:
10       double length_; // Nothing!
11};
```

**Derived Class: Square**

```cpp
// Square.h
1 #pragma once
2 #include "Shape.h"
3 class Square {
4 public:
5    double getArea() const;
6 private:
7       // Nothing!
8}; // Nothing!
```

In the above code, Square is derived from the base class Shape:
- All **public** functionality of Shape is part of Square:
  ```cpp
  main.cpp
5 int main() {
6     Square sq;
7     sq.getLength(); // Returns 1, the len init’d by Shape’s default ctor
8 ... // Returns 1, the len init’d by Shape’s default ctor
  ```
- [Private Members of Shape]:

**Calling Base Class Constructors (Initializer List!)**

```cpp
// Square.h
6 public:
7    Square(double length);
```

```cpp
// Square.cpp
6 Square::Square(double length) : Shape(length) {} // No print_1()
```

**Functions: virtual and pure virtual**
- The **virtual** keyword:

```cpp
// Cube.cpp
6 Cube::print_1() {
7     cout << "Cube" << endl;
8 }
```

```cpp
// RubikCube.cpp
6 RubikCube::print_1() // No print_1()
```

```cpp
// Cube.cpp
6 Cube::print_2() {
7     cout << "Cube" << endl;
8 }
```

```cpp
// RubikCube.cpp
6 RubikCube::print_2() // No print_3()
```

```cpp
// Cube.cpp
6 Cube::print_3() {
7     cout << "Cube" << endl;
8 }
```

```cpp
// RubikCube.cpp
6 RubikCube::print_3() // No print_3()
```

```cpp
// Cube.cpp
6 Cube::print_4() {
7     cout << "Cube" << endl;
8 }
```

```cpp
// RubikCube.cpp
6 RubikCube::print_4() // No print_3()
```

```cpp
// Cube.cpp
6 Cube::print_5() {
7     cout << "Cube" << endl;
8 }
```

```cpp
// RubikCube.cpp
6 RubikCube::print_5() {
7     cout << "Rubik" << endl;
8 }
```

```cpp
// Cube.cpp
6 Cube &c = rc; // Cube &c = rc;
```

CS 225 – Things To Be Doing:
1. Theory Exam #1 starts tomorrow!
2. lab_memory this week in labs (due Sunday)
3. MP2 released (EC due Monday)
4. Daily POTDs every M-F for daily extra credit!