#7: Inheritance
September 12, 2018 · Wade Fagen-Ulmschneider

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

**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>Assignment operator</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.
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
class Shape {
public:
    Shape();
    Shape(double length);
    double getLength() const;
private:
    double l_length_;  // Nothing!
};
```

**Derived Class: Square**

```cpp
#include "Shape.h"

class Square {
public:
    double getArea() const;
private:
    // Nothing!
};
```

In the above code, **Square** is derived from the base class **Shape**:

- **All public functionality of Shape** is part of **Square**:

```cpp
int main() {
    Square sq;
    sq.getLength();  // Returns 1, the length 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);

Square.cpp
6 | Square::Square(double length) : Shape(length) {}  // In .h file:
   | virtual Cube::print_5() = 0;
```

**Functions: virtual and pure virtual**

- **The virtual keyword**:  

```cpp
Cube.cpp
RubikCube.cpp
Cube::print_1() {  
    cout << "Cube" << endl;
}

Cube::print_2() {  
    cout << "Cube" << endl;
}

virtual Cube::print_3() {  
    cout << "Cube" << endl;
}

// In .h file:
virtual Cube::print_5() = 0;

RubikCube::print_2() {  
    cout << "Rubik" << endl;
}

virtual RubikCube::print_4() {  
    cout << "Rubik" << endl;
}

RubikCube::print_4() {  
    cout << "Rubik" << endl;
}

RubikCube::print_5() {  
    cout << "Rubik" << endl;
}

RubikCube rb;
RubikCube &c = rb;
```

**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!