#7: Inheritance

September 11, 2019 · *G Carl Evans*

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:

Cube.h

```
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:

Arithmetic	`+   -   *   /   %   ++   --`
Bitwise	`&   |   ^   ~   <<   >>`
Assignment	`=`
Comparison	`==   !=   >   <   >=   <=`
Logical	`!   &&   ||`
Other	`[]   ()   ->`

General Syntax:

Adding overloaded operators to Cube:

Cube.h

```
#pragma once

class Cube {
  public:
    // ...





    // ...
```

Cube.cpp

```
/* ... */









/* ... */
```

One Very Powerful Operator: Assignment Operator

Cube.h

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

Cube.cpp

```
Cube & Cube::operator=(const Cube & other) { ... }
```

Functionality Table:

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

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`

Shape.h

```
class Shape {
  public:
    Shape();
    Shape(double length);
    double getLength() const;

  private:
    double length_;
};
```

Derived Class: `Square`

Square.h

```
#pragma once

#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`:

main.cpp

```
int main() {
  Square sq;
  sq.getLength(); // Returns 1, the len init'd
                  // by Shape's default ctor
  ...
```

- [Private Members of `Shape`]:

Calling Base Class Constructors *(Initializer List!)*

Square.h

```
  public:
    Square(double length);
```

Square.cpp

```
Square::Square(double length) : Shape(length) { }
```

Functions: virtual and pure virtual

- The **virtual** keyword:

Cube.cpp

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

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

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

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

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

RubikCube.cpp

```
// No print_1()



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

// No print_3()



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

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

	`Cube c;`	`RubikCube c;`	`RubikCube rc;`
			`Cube &c = rc;`

`c.print_1();`			
`c.print_2();`			
`c.print_3();`			
`c.print_4();`			
`c.print_5();`			

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!