

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	
5	class Cube {
6	public:
7	Cube(); // default ctor
8	Cube(double length); // 1-param ctor
9	Cube(const Cube & other); // custom copy ctor
10	~Cube(); // destructor, or dtor
11	...

...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		Cube.cpp	
1	#pragma once	...	/* ... */
2		40	
3	class Cube {	41	
4	public:	42	
...	// ...	43	
10		44	
11		45	
12		46	
13		47	
14		48	
...	//	/* ... */

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	
4	class Shape {
5	public:
6	Shape();
7	Shape(double length);
8	double getLength() const;
9	
10	private:
11	double length_;
12	};

Derived Class: Square

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	};

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

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

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

- [Private Members of **Shape**]:

Calling Base Class Constructors (*Initializer List!*)

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

Square.cpp	
6	Square::Square(double length) : Shape(length) { }

Functions: virtual and pure virtual

- The **virtual** keyword:

Cube.cpp	RubikCube.cpp
Cube::print_1() { cout << "Cube" << endl; }	// No print_1()
Cube::print_2() { cout << "Cube" << endl; }	RubikCube::print_2() { cout << "Rubik" << endl; }
virtual Cube::print_3() { cout << "Cube" << endl; }	// No print_3()
virtual Cube::print_4() { cout << "Cube" << endl; }	RubikCube::print_4() { cout << "Rubik" << endl; }
// In .h file: virtual Cube::print_5() = 0;	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:

- Theory Exam #1 starts tomorrow!
- lab_memory this week in labs (*due Sunday*)
- MP2 released (*EC due Monday*)
- Daily POTDs every M-F for daily extra credit!