#6: Lifecycle of Classes

February 3, 2020 · *G Carl Evans*

Copy Constructor

When a non-primitive variable is passed/returned **by value,** a copy must be made. As with a constructor, an automatic copy constructor is provided for you if you choose not to define one:

All **copy constructors** will:

The **automatic copy constructor**:

1.

2.

To define a **custom copy constructor**:

cs225/Cube.h

```
class Cube {
  public:
    Cube();                 // default ctor
    Cube(double length);  // 1-param ctor


    double getVolume();
    double getSurfaceArea();

  private:
    double length_;
};
```

Recall the joinCubes function:

joinCubes-{byValue,byReference,byPointer}.cpp

```
Cube joinCubes(Cube ___ c1, Cube ___ c2) {
  double totalVolume = c1.getVolume() + c2.getVolume();

  double newLength = std::pow( totalVolume, 1.0/3.0 );

  Cube result(newLength);
  return result;
}
```

Bringing Concepts Together:

How many times do our different joinCubes files call each constructor?

	By Value	By Pointer	By Reference
`Cube()`			
`Cube(double)`			
`Cube(const Cube &)`			

Cubes Unite!

Consider a Tower made of three Cubes:

Tower.h

```
#pragma once

#include "cs225/Cube.h"
using cs225::Cube;

class Tower {
  public:
    Tower(Cube c, Cube *ptr, const Cube &ref);
    Tower(const Tower & other);

  private:
    Cube cube_;
    Cube *ptr_;
    const Cube &ref;
};
```

Automatic Copy Constructor Behavior:

The behavior of the automatic copy constructor is to make a copy of every variable. We can mimic this behavior in our Tower class:

Tower.cpp

```
Tower::Tower(const Tower & other) {
  cube_ = other.cube_;
  ptr_ = other.ptr_;
  ref_ = other.ref_;
}
```

```
Tower::Tower(const Tower & other) : cube_(other.cube_),
   ptr_(other.ptr_), ref_(other.ref_) { }
```

...we refer to this as a ______________________________ because:

Deep Copy via Custom Copy Constructor:

Alternatively, a custom copy constructor can perform a deep copy:

```
Tower.cpp
Tower::Tower(const Tower & other) {
  // Deep copy cube_:



  // Deep copy ptr_:



  // Deep copy ref_:



}
```

Destructor

The <u>last and final</u> member function called in the lifecycle of a class is the destructor.

Purpose of a **destructor**:

The **automatic destructor**:

1.

2.

Custom Destructor:

```
cs225/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
    ...
```

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:

```
cs225/Cube.h
#pragma once

class Cube {
  public:
    // ...





    // ...
```

```
cs225/Cube.cpp
/* ... */









/* ... */
```

Assignment Operator

Among all of the operators, one the assignment operator is unique:

1.

2.

CS 225 – Things To Be Doing:

1. Quiz 1 finishes today.
2. MP1 due Wednesday@ 11:59pm
3. MP2 released on Tuesday *(start early for extra credit!)*
4. Lab Attendance in your registered lab section
5. Daily POTDs every M-F for daily extra credit!