

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
4 class Cube {
5     public:
6         Cube();           // default ctor
7         Cube(double length); // 1-param ctor
8
9
10        double getVolume();
11        double getSurfaceArea();
12
13    private:
14        double length_;
15 };
    
```

Recall the **joinCubes** function:

```

joinCubes-{byValue,byReference,byPointer}.cpp
15 Cube joinCubes(Cube c1, Cube c2) {
16     double totalVolume = c1.getVolume() + c2.getVolume();
17
18     double newLength = std::pow( totalVolume, 1.0/3.0 );
19
20     Cube result(newLength);
21     return result;
22 }
    
```

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
1 #pragma once
2
3 #include "cs225/Cube.h"
4 using cs225::Cube;
5
6 class Tower {
7     public:
8         Tower(Cube c, Cube *ptr, const Cube &ref);
9         Tower(const Tower & other);
10
11    private:
12        Cube cube_;
13        Cube *ptr_;
14        const Cube &ref;
15 };
    
```

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
10 Tower::Tower(const Tower & other) {
11     cube_ = other.cube_;
12     ptr_ = other.ptr_;
13     ref_ = other.ref_;
14 }
10 Tower::Tower(const Tower & other) : cube_(other.cube_),
11     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
11 Tower::Tower(const Tower & other) {
12     // Deep copy cube_ :
13
14
15
16     // Deep copy ptr_ :
17
18
19
20     // Deep copy ref_ :
21
22
23 }
```

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.
- 2.

Custom Destructor:

```
cs225/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        ...
```

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		cs225/Cube.cpp	
1	#pragma once	...	/* ... */
2		10	
3	class Cube {	11	
4	public:	12	
...	// ...	13	
16		14	
17		15	
18		16	
19		17	
20		18	
...	//	/* ... */

Assignment Operator

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

- 1.
- 2.

CS 225 – Things To Be Doing:

1. Exam 1 next Friday.
2. mp_intro, lab_intro, lab_debug all due Wednesday@ 11:59pm
3. MP2 released on Tuesday (*start early for extra credit!*)
4. Daily POTDs every M-F for daily extra credit!