

Returning from a function

Identical to passing into a function, we also have three choices on how memory is used when returning from a function:

Return by value:

```
15 Cube joinCubes(const Cube &s1, const Cube &s2)
```

Return by reference:

```
15 Cube &joinCubes(const Cube &s1, const Cube &s2)
```

...remember: never return a reference to stack memory!

Return by pointer:

```
15 Cube *joinCubes(const Cube &s1, const Cube &s2)
```

...remember: never return a reference to stack memory!

Copy Constructor

When a non-primitive variable is passed/returned **by value**, a copy must be made.

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};
```

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. 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		
Copy 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.

The Rule of Zero

CS 225 and Rule Three/Five/Zero

In CS 225 We will:

CS 225 – Things To Be Doing:

1. Finish lab_intro
2. Start on mp_stickers
3. First PotD released today due Monday.