#6: Lifecycle of Classes

January 29, 2018 · *Wade Fagen-Ulmschneider*

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:

```
Sphere  joinSpheres(const Sphere &s1, const Sphere &s2)
```

Return by reference:

```
Sphere &joinSpheres(const Sphere &s1, const Sphere &s2)
```

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

Return by pointer:

```
Sphere *joinSpheres(const Sphere &s1, const Sphere &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. 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**:

sphere.h

```
class Sphere {
  public:
    Sphere();          // default ctor
    Sphere(double r);  // 1-param ctor
    Sphere(const Sphere & other);    // custom copy ctor
    ...
```

Bringing Concepts Together:

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

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

joinSpheres-{byValue,byReference,byPointer}.cpp

```
Sphere joinSpheres(Sphere ___ s1, Sphere ___ s2) {
  double totalVolume = s1.getVolume() + s2.getVolume();

  double newRadius = std::pow(
    (3.0 * totalVolume) / (4.0 * 3.141592654),
    1.0/3.0
  );

  Sphere result(newRadius);

  return result;
}
```

A Sphere, A Universe.

Consider a Universe of three Spheres:

Universe.h

```
#ifndef UNIVERSE_H_
#deifne UNIVERSE_H_

#include "Sphere.h"
using namespace cs225;

class Universe {
  public:
    Universe();        // default ctor
    Universe(Sphere s, Sphere *q, Sphere &r); // 3-param
    Universe(const Universe & other);
    // …
  private:
    Sphere p_, *q_, &r;
};

#endif
```

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 Universe class:

Universe.cpp
```
Universe::Universe(const Universe & other) {
  p_ = other.p_;
  q_ = other.q_;
  r_ = other.r_;
}
```

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

Deep Copy via Custom Copy Constructor:
Alternatively, a custom copy constructor can perform a deep copy:

Universe.cpp
```
Universe::Universe(const Universe & other) {
  // Deep copy p_:



  // Deep copy q_:



  // Deep copy r_:



}
```

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:

sphere.h
```
class Sphere {
  public:
    Sphere();            // default ctor
    Sphere(double r);    // 1-param ctor
    Sphere(const Sphere & other);    // custom copy ctor
    ~Sphere();           // 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 Sphere:

sphere.h
```
#ifndef SPHERE_H
#define SPHERE_H

class Sphere {
  public:
    // ...




    // ...
```

sphere.cpp
```
/* ... */









/* ... */
```

CS 225 – Things To Be Doing:

1. Theory Exam #1 Starts Tomorrow (Register in the CBTF)
2. MP1 due tonight; grace period until Tuesday @ 11:59pm
3. MP2 released on Tuesday *(start early for extra credit!)*
4. Lab Extra Credit → Attendance in your registered lab section!
5. Daily POTDs every M-F for daily extra credit!