Today's announcements

- MP2, due on 02/09, 11:59pm

- lab_gdb release is to be announced on piazza

- First in-lab exam, Feb 10, 11, 12!
  Make sure to come to the lab to which you are registered, only then you will be graded!

- Exam materials:
  https://chara.cs.illinois.edu/cs225/exams/mt1/

Where were we? Constructors, Destructors and Copy Constructors

Write the **copy constructor** function signature as it appears in sphere class definition:

List two instances in which a class's **copy constructor** is called:

1.

2.

Write the **destructor** function signature as it appears in sphere class definition:

List two instances in which a class's **destructor** is called:

1.

2.

The destructor, a summary:

1. Destructor is never “called.” Rather, we provide it for the system to use in two situations:

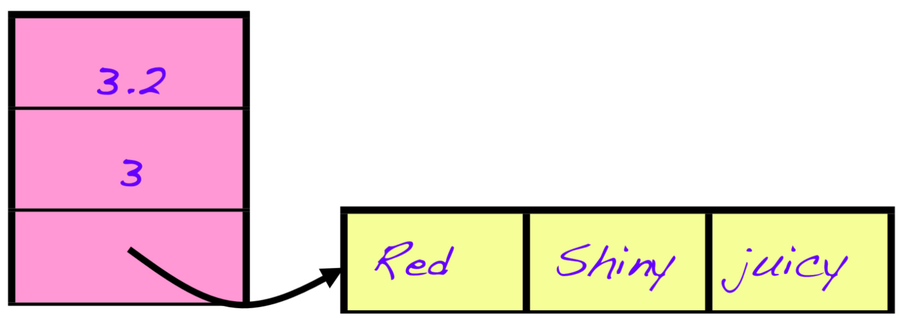
   a) _____

   b) _____

2. If your constructor, _____, allocates dynamic memory, then you need a destructor.

3. Destructor typically consists of a sequence of delete statements.

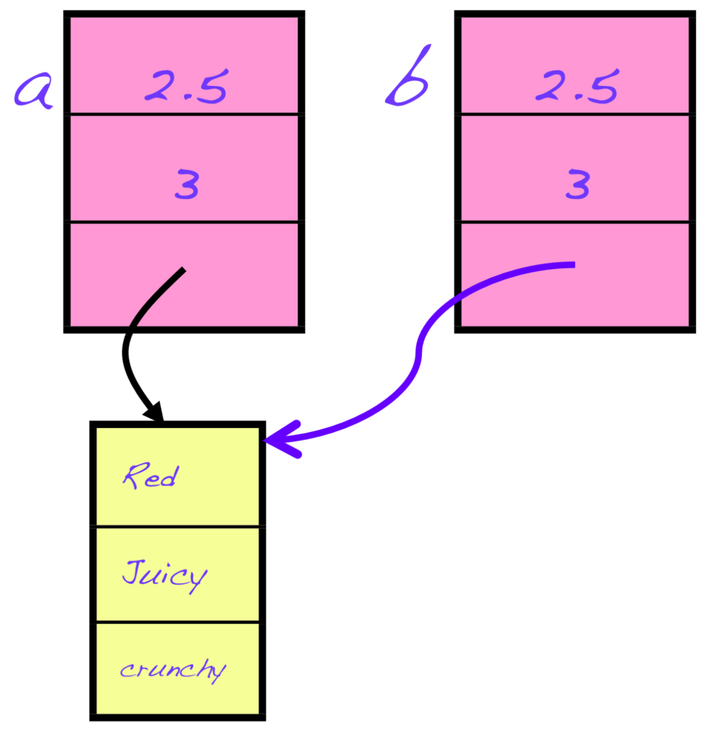
```
class sphere{
public:
   //tons of other stuff
   ~sphere();
private:
   double theRadius;
   int numAtts;
   string * atts;
};
```



One more problem: *default assignment is memberwise, so we "redefine" or "overload" =.*

```
class sphere{

public:

sphere();

sphere(double r);

sphere(const sphere & orig);

~sphere();

_________ operator=(_____________________);

…

private:

double theRadius;

int numAtts;

string * atts;

};
```

```
int main(){
   sphere a, b;
   // initialize a
   b = a;
   return 0;
}
```



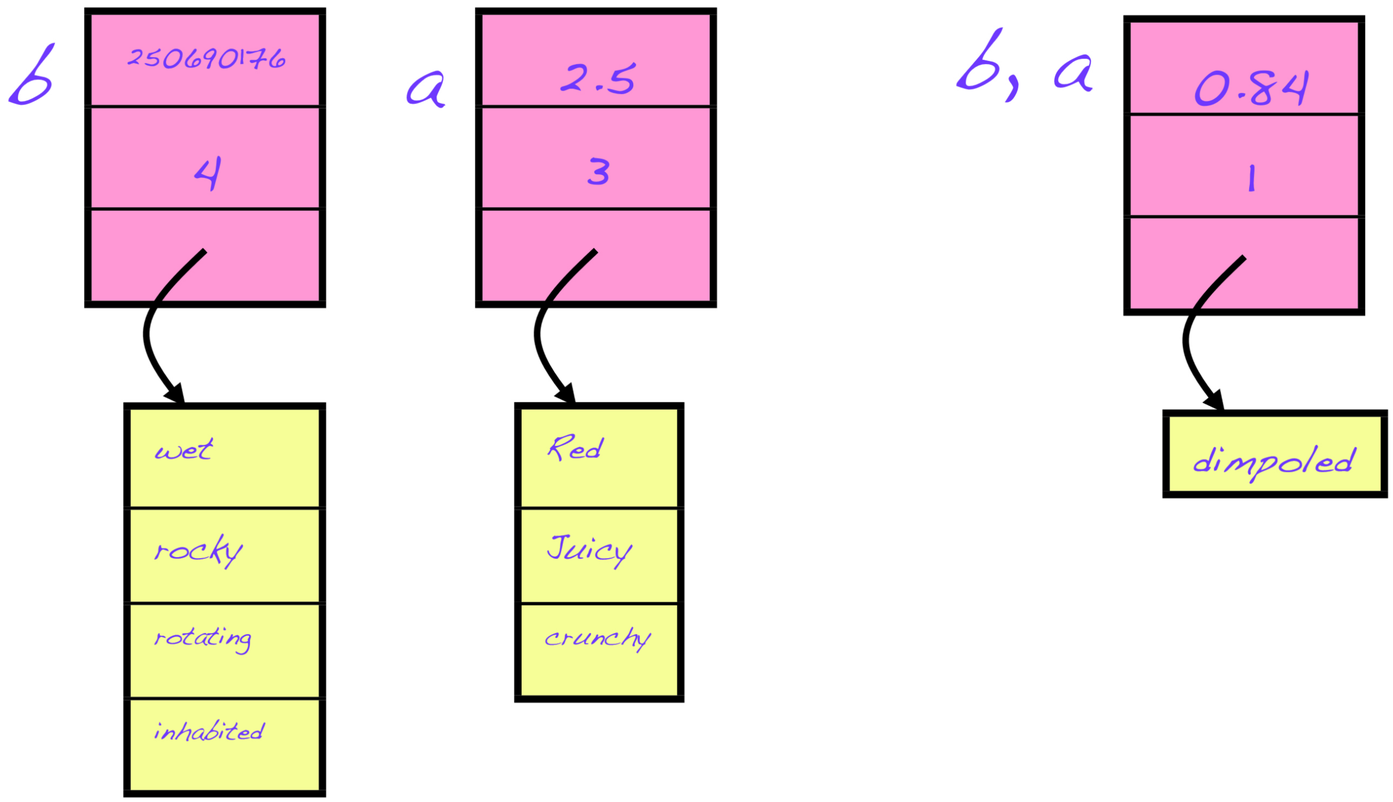
Overloaded operators:

```
int main(){
   // declare a,b,c









   // initialize a,b
   c = a + b;
   return 0;
}
```

```
// overloaded operator
sphere & sphere::operator+
    (const sphere & s){




}
```

Some things to think about...

b=a



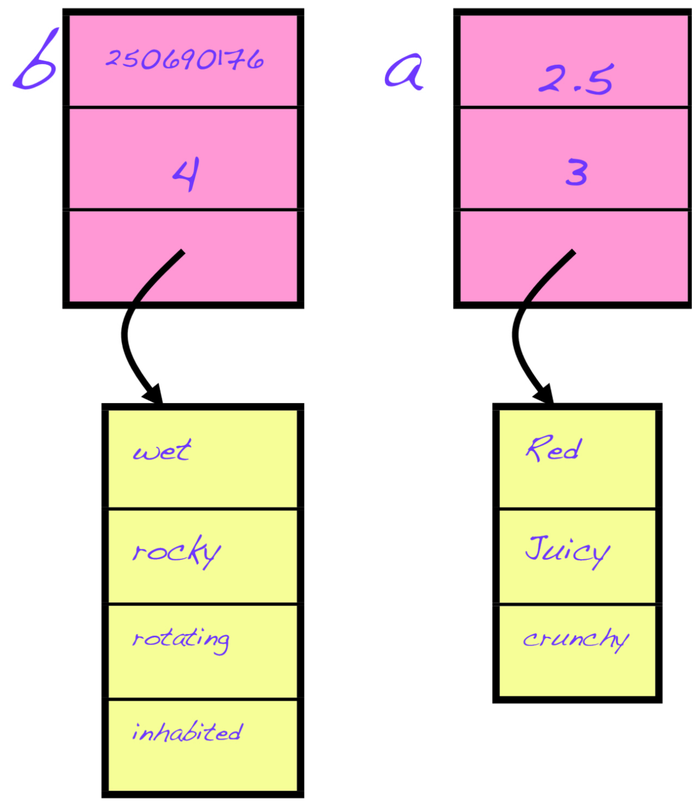
c=b=a

Operator=:

```
…
// overloaded =
sphere & sphere::operator=(const sphere & rhs){
  }
…
```

```
c
p
s
s
s
~
…
p
d
int numAtts;

string * attributes;

};
```

```
int main(){
  sphere a, b;
   // initialize a
   b = a;
   return 0;
}
```



The Rule of the Big Three:

If you have a reason to implement any one of

- _____
- _____
- _____

then you must implement all three.

Object Oriented Programming

Three fundamental characteristics:

encapsulation - separating an object's data and implementation from its interface.

inheritance -

polymorphism - a function can behave differently, depending on the type of the calling object.