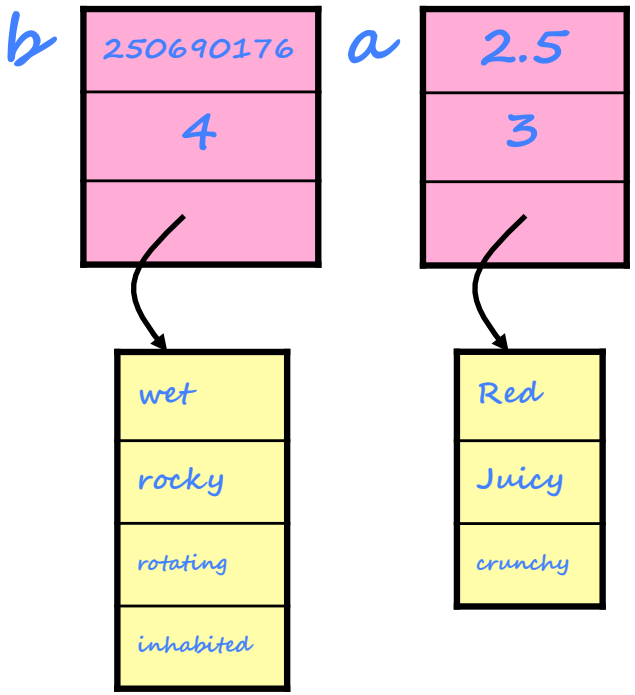
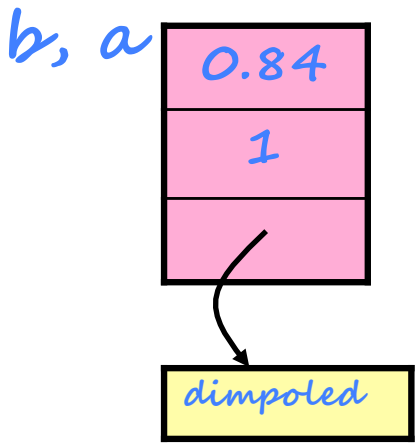
Announcements

MP2 available, due 9/15, 11:59p.

b=a



b=a



c=b=a

Operator=, the plan:

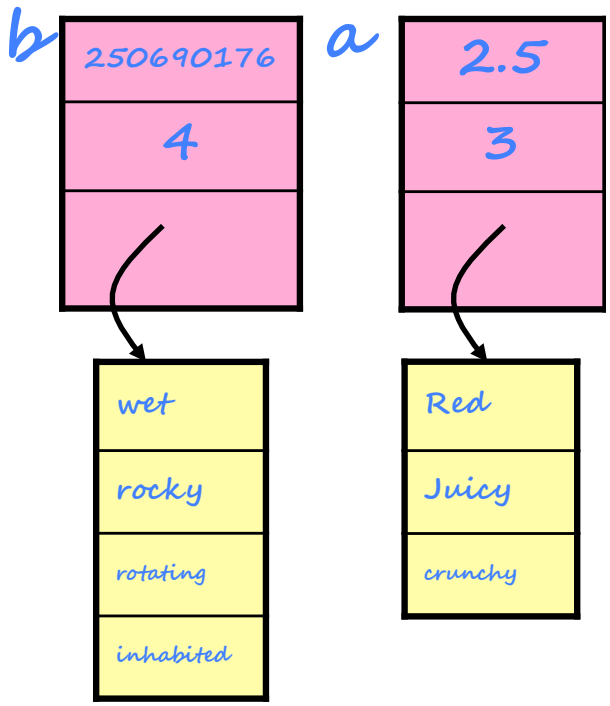
```
…
// overloaded =
sphere & sphere::operator=(const sphere & rhs){

}
```

```
c
p
sp
sp
sp
~s

…
p
dd
int numAtts;
string * attributes;
};
```

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



Operator=:

```
…
// overloaded =
sphere & sphere::operator=(const sphere & rhs){
   //protect against re-assignment


      //clear lhs



      //copy rhs




   //return a helpful value

}
…
```

Operator=:

```
class sphere{

public:

sphere();

sphere(double r);

sphere(const sphere & o

~sphere();

sphere & operator=(cons…

private:

double theRadius;

int numAtts;

string * attributes;

};
```

```
…

sphere & sphere::operator=(const sphere & rhs){

   if (this != &rhs) {

      clear();
      copy(rhs);

   }
   return *this;

}
```

Why not `(*this != rhs)` ?

-
-
-

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.

Inheritance: a simple first example

```
class sphere {

public:

sphere();

sphere(double r);

double getVolume();

void setRadius(double r);

void display();

private:

double theRadius;



};
```

```
class ball:public sphere {

public:

ball();

ball(double r string n);

string getName();

void setName(string n);

void display();

private:

string name;

};
```

inheritance rules:

-
-
-

```
int main() {
sphere a;
cout << a.surfaceArea;}
```

Subclass substitution (via examples):

```
void printVolume(sphere  t){
   cout << t.getVolume()  << endl;}

int main()  {
   sphere s(8.0);
   ball b(3.2, "pompom");

   double a = b.getVolume();

   printVolume(s);
   printVolume(b);
}
```

```
Base b;
Derived d;

b=d;

d=b;
```

```
Base * b;
Derived * d;

b=d;

d=b;
```

something to consider:

```
class sphere {
public:
    sphere();
    sphere(double r);
    …
    void display();
private:
    double theRadius;
};
```

```
void sphere::display() {
   cout << "sphere" << endl;
}
```

```
class ball:public sphere {
public:
    ball();
    ball(double r string n);
    …
    void display();
private:
    string name;
};
```

```
void ball::display() {
   cout << "ball" << endl;
}
```

ex1

```
sphere s;
ball b;
s.display();
b.display();
```

ex2

```
sphere * sptr;
sptr = &s;
sptr->display();
```

ex3

```
sphere * sptr;
sptr = &b;
sptr->display();
```