Announcements

Course policies:

http://cs.illinois.edu/class/cs225

For general assistance:

http://piazza.com/class#spring2013/cs225

MP2 available, due 2/5, 11:59p. EC: 1/29, 11:59p.

Copy constructor - a function you write for the system to use.

```
class sphere{
public:
sphere();
sphere(double r);
sphere(const sphere & orig);
void setRadius(double newRad);
double getDiameter() const;
…
private:
double theRadius;
int numAtts;
string * atts;
};
```

Use 1:

```
sphere myFun(sphere s){
   //play with s
   return s;
}

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

Use 2:

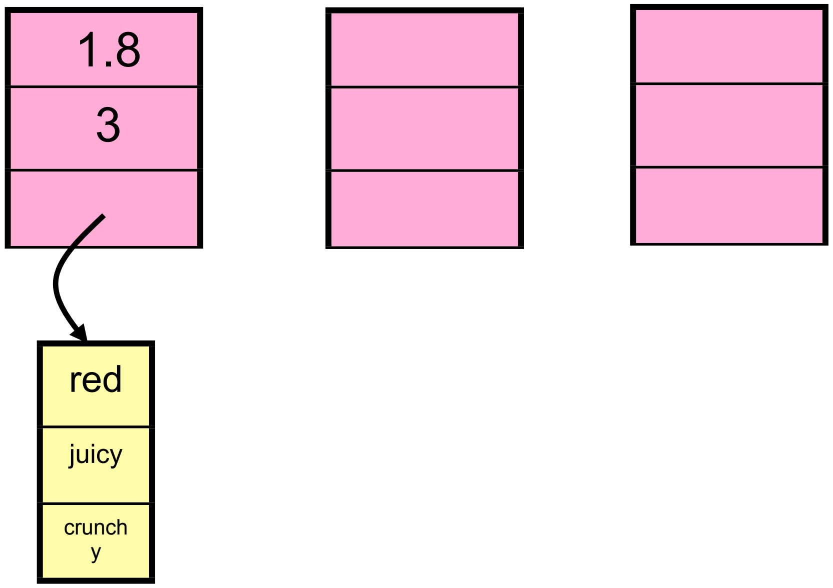
```
int main(){
  sphere a;

  sphere c =

};
```

Use 3? *upon return...sometimes*

Copy constructor:

```
…
//copy constructor
sphere::sphere(const sphere & orig)
{



}
…
```

```
class sphere{
public:
sphere();
sphere(double r);
sphere(const sphere & orig);
void setRadius(double newRad);
double getDiameter() const;
…
private:
double theRadius;
int numAtts;
string * atts;
};
```

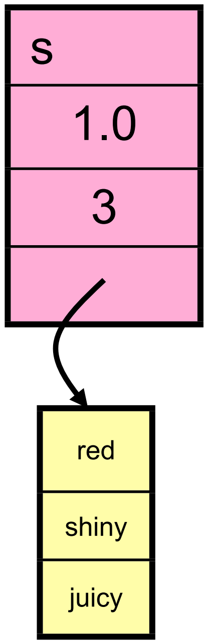


Poser: cctor - why pbr?

```
class sphere{
public:
sphere();
sphere(double r);
sphere(const sphere & orig);
void setRadius(double newRad);
double getDiameter() const;
…
private:
double theRadius;
int numAtts;
string * atts;
};
```

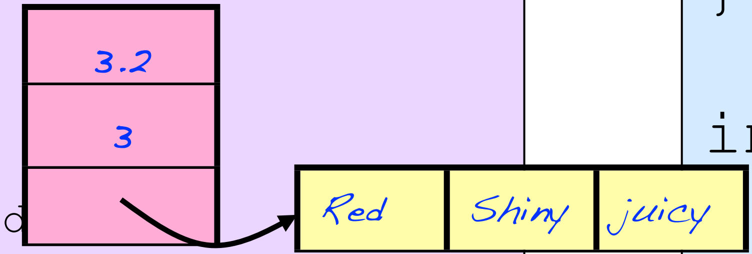
```
…
//copy constructor
sphere::sphere(const sphere & orig):
theRadius(orig.theRadius),numatts(orig.numAtts)
{
  atts = new string[numAtts];
  for(int i=0; i<numAtts;i++)
    atts[i]= orig.atts[i];
}
…
```

```
int main(){
  sphere s;
   …// initialize s
   sphere t(s); //invokes CC
   return 0;
}
```



Destructors:

```
class sphere{

public:

sphere();

sphere(double r);

sphere(const sphere & orig);

~sphere();

…

private:

double theRad

int numAtts;

string * atts;

};
```



```
//destructor
sphere::~sphere(){



}
```

```
void myFun(sphere s){
    sphere t(s);
    …
    // play with s and t
    …
}


int main(){

    sphere a;
    myFun(a);

    sphere * b = new sphere;
    delete b;
    return 0;
}
```

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.

Quiz: Name two different situation in which the copy constructor is invoked by the system.

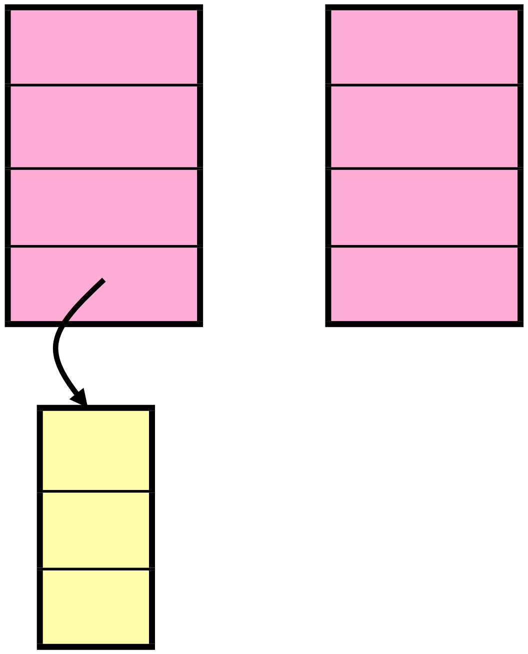
1.

2.

One more problem:

```
class sphere{
public:
sphere();
sphere(double r);
sphere(const sphere & orig);
~sphere();
…
private:
double theRadius;
int numAtts;
string * atts;
};
```

```
int main(){
   sphere a, b;
   // change b somehow
   a = b;
   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){






}
```

Overloaded operators: what can be overloaded?

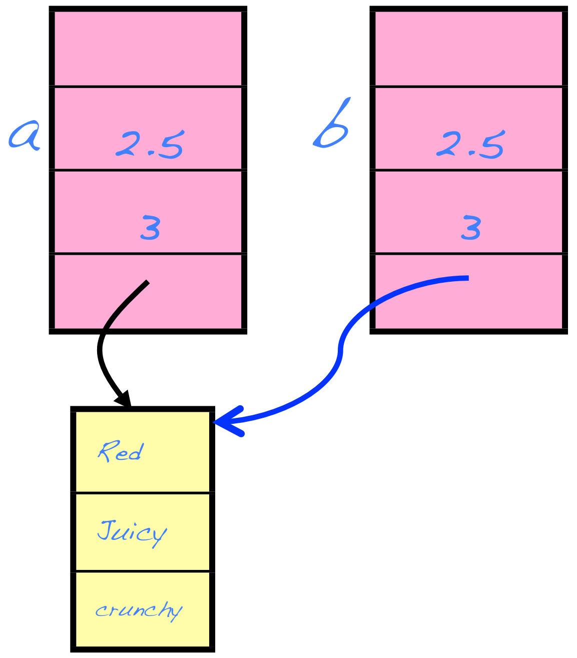
arithmetic operators, logical operators, I/O stream operators

```
    +     -     *     /     =     <     >     +=    -=    *=    /=
  <<   >>     <<=   >>=   ==    !=    <=    >=    ++    --    %     &
         ^     !     | ~     &=    ^=    |=    &&    ||    %=
                      []    ()    ,     ->*   ->
               new   delete    new[]      delete[]
```

One more problem: default assignment is memberwise, so we redefine =.

```
class sphere{
public:
sphere();
sphere(double r);
sphere(const sphere & orig);
~sphere();
…
private:
double theRadius;
int numAtts;
string * atts;
};
```

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

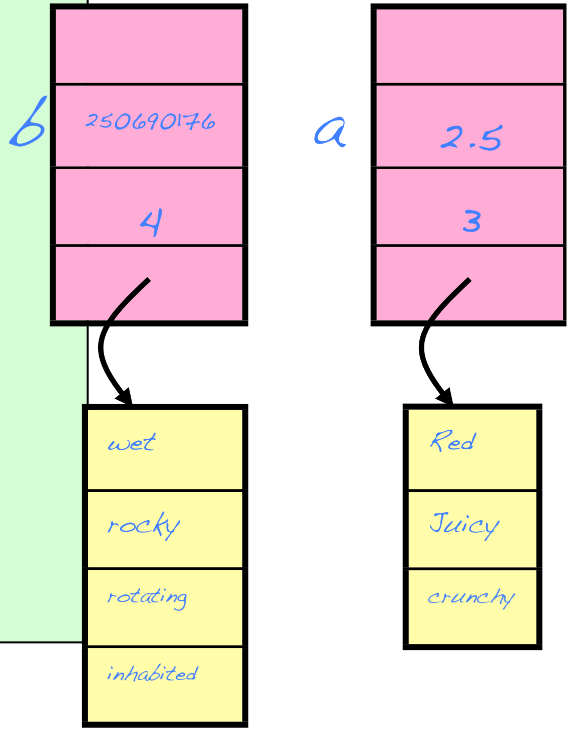


Operator= the plan:

```
clas

publ

sphere
sphere
sphere
~spher

…

priv
double theRadius;
int numAtts;
string * attributes;

};
```

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






}
…
```

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



Operator=:

```
class sphere{
public:
sphere();
sphere(double r);
sphere(const sphere & 
~sphere();

…
private:
double theRadius;
int numAtts;
string * attributes;
};
```

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


      //clear lhs



      //copy rhs




   //return a helpful value

}
…
```

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