

Announcements

MP2 available, due 9/13, 11:59p.

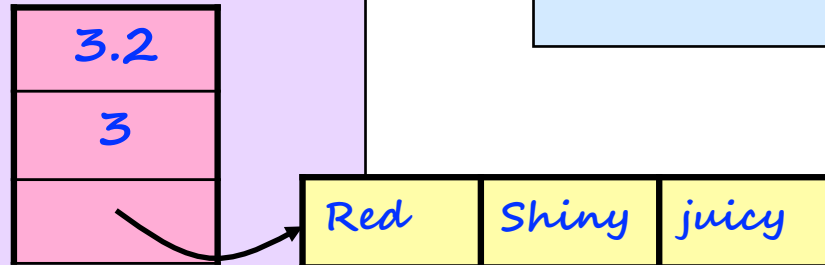
```
class sphere{  
public:  
    sphere();  
    sphere(const sphere & orig);  
    ...  
private:  
    double theRadius;  
    int numAtts;  
    string * atts;  
};
```

Copy Constructor discussion:

1. Why is the ctor's param pbr?
2. What does it mean that the ctor's param is const?
3. Why did we need to write a custom ctor?

Destructors:

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



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

```
int main() {
    sphere a;
    myFun(a);
}
```

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

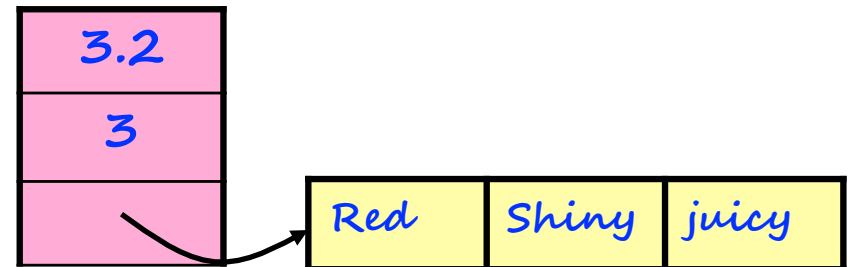
}
```

Destructors:

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

private:
    double theRadius;
    int numAtts;
    string * atts;
};
```

```
int main() {
    sphere * b = new sphere;
    delete b;
    return 0;
}
```



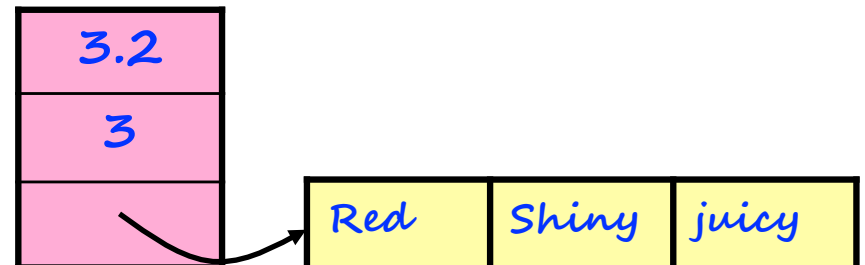
```
//destructor
sphere::~~sphere() {

}
```

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.

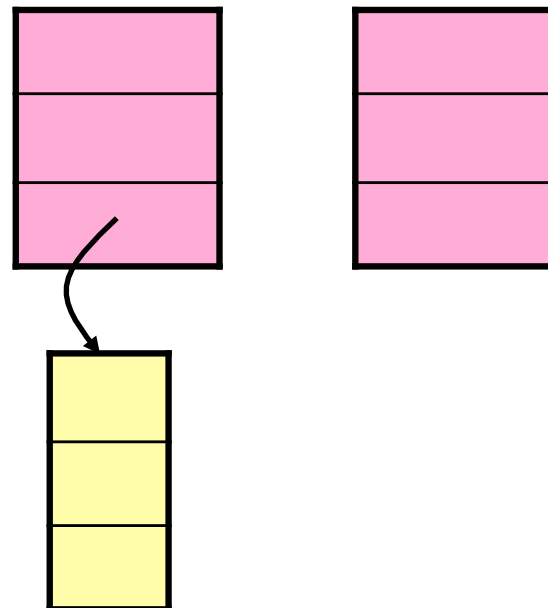
```
class sphere{  
public:  
    //tons of other stuff  
    ~sphere();  
private:  
    double theRadius;  
    int numAtts;  
    string * atts;  
};
```



One more problem:

```
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  
    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){

}
```

Overloaded operators: what can be overloaded?

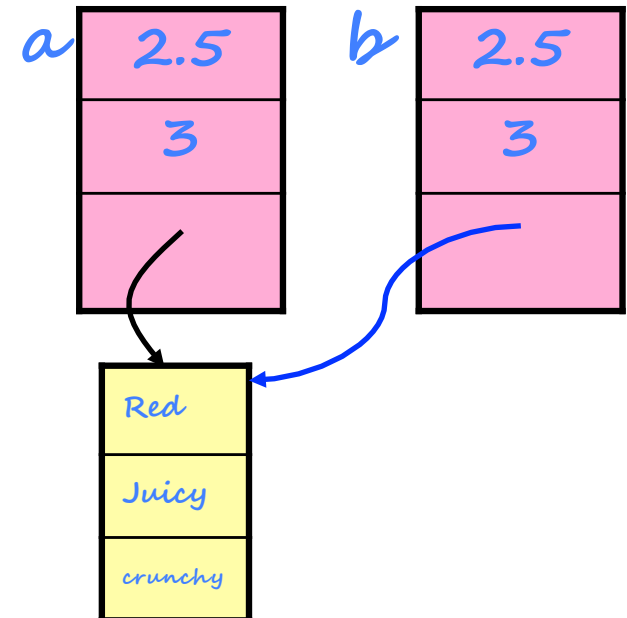
arithmetic operators, logical operators, I/O stream operators

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

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

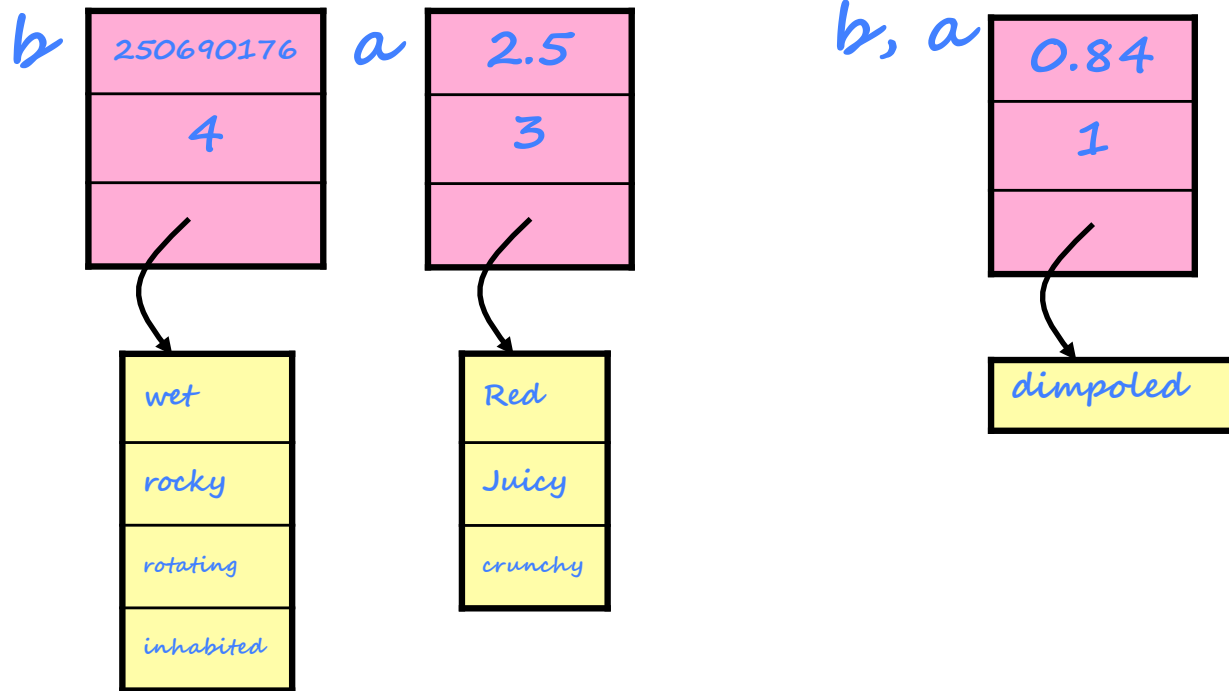
```
class sphere{  
public:  
    sphere();  
    sphere(double r);  
    sphere(const sphere & orig);  
    ~sphere();  
    _____ operator=( _____ );  
    ...  
private:  
    double theRadius;  
    int numAtts;  
    string * atts;  
};
```

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



Some things to think about...

$b=a$



$c=b=a$

Operator=, the plan:

```
...
// overloaded =
C sphere & sphere::operator=(const sphere & rhs){
p
sp
sp
sp
~s
...
p
de}

int numAtts;
string * attributes;
};

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

