Lab 8 - The Big 3

OH, HI; I'M HERE FROM THE INTERNET.
WHAT ARE YOU DOING!?
GLUING CAPTIONS TO YOUR CATS.
RRRR

ReadMe

Reminders

- No lab next week
- Lab 8 due Sunday, November 18th at 8pm
- Project 4 due Monday, November 19th at 8pm

Agenda

- Review - The Big Three
- Worksheet
- Lobster Exercise
- Lab 8
- Project Questions?

Dynamic Memory

Recall:

If we use dynamic memory, we have a responsibility to clean up that memory when we're done!

Example:

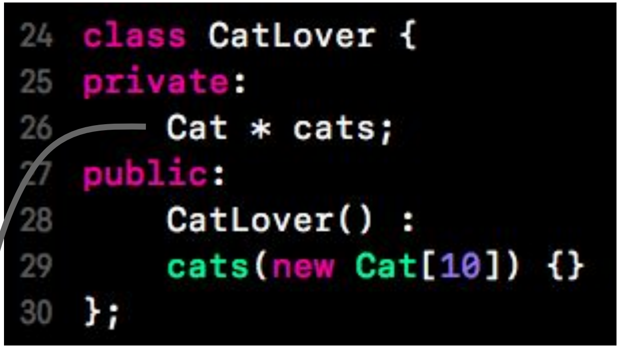
```
int num[] = new int[5];

delete[] num;
```

OR

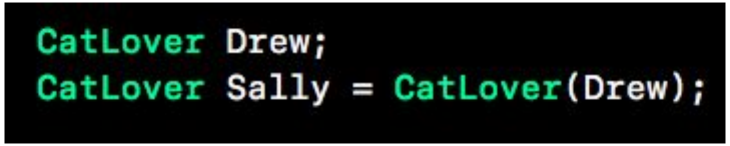
```
int num = new int(5);

delete num;
```

Remember to use the correct syntax for deleting an array of items vs. a single item!

What happens when a class manages dynamic memory?

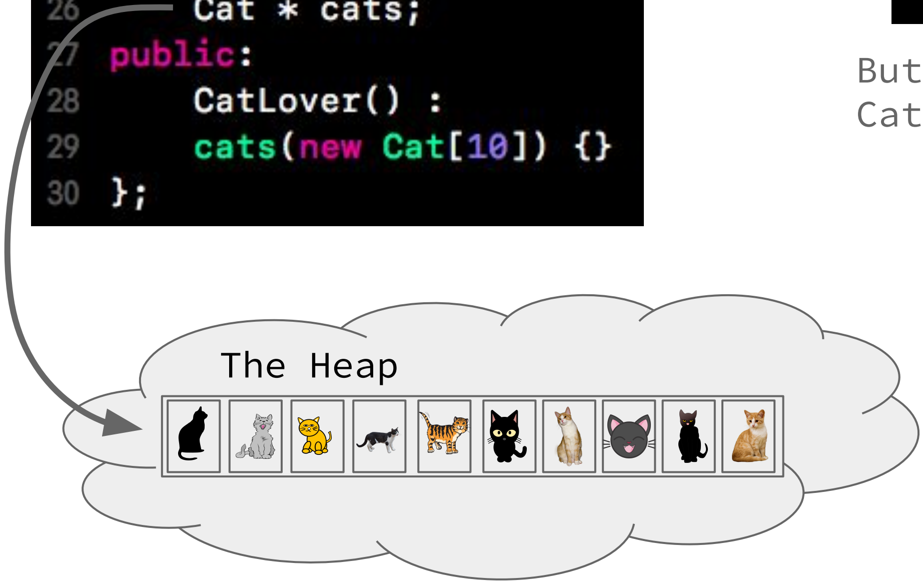


```
class CatLover {
private:
    Cat * cats;
public:
    CatLover() :
    cats(new Cat[10]) {}
};
```



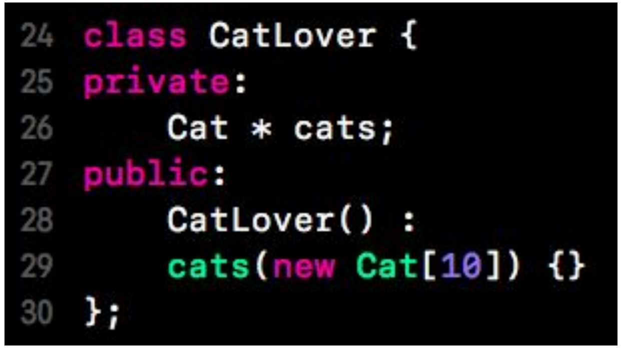
```
CatLover Drew;
CatLover Sally = CatLover(Drew);
```

But what happens when we set Sally = CatLover(Drew)?

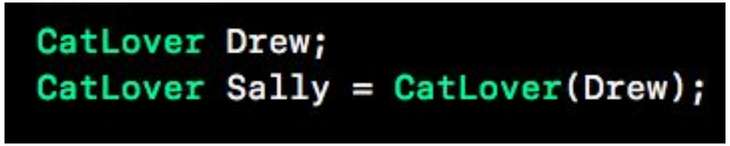


When this constructor runs, “cats” points to a new array of Cats on the heap

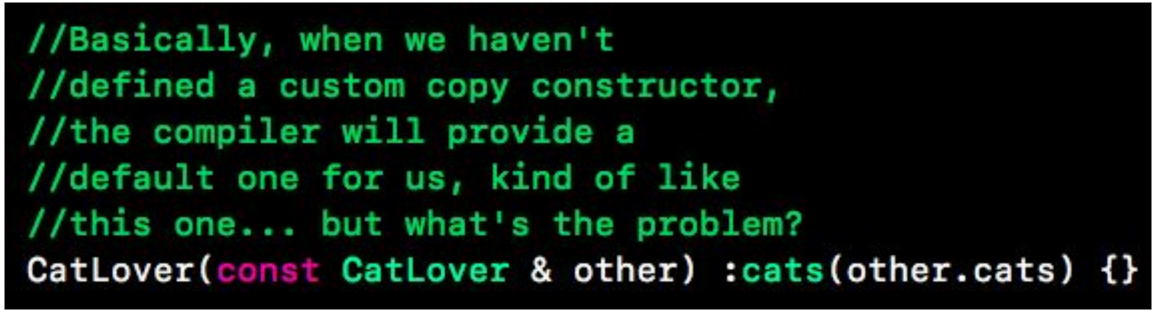
What happens when a class manages dynamic memory?



```cpp
class CatLover {
private:
    Cat * cats;
public:
    CatLover() :
    cats(new Cat[10]) {}
};
```

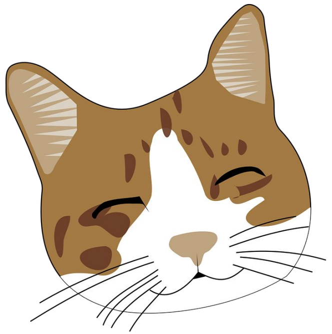


```cpp
CatLover Drew;
CatLover Sally = CatLover(Drew);
```

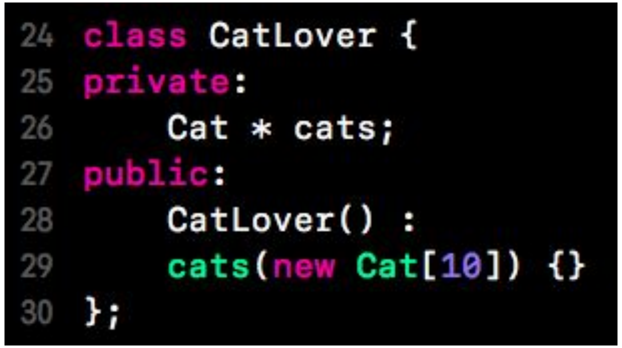


```cpp
//Basically, when we haven't
//defined a custom copy constructor,
//the compiler will provide a
//default one for us, kind of like
//this one... but what's the problem?
CatLover(const CatLover & other) :cats(other.cats) {}
```

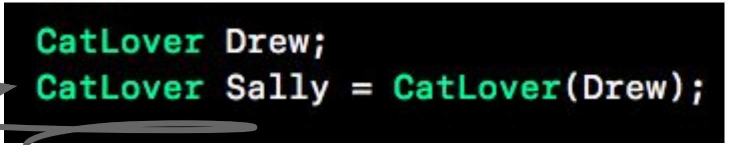
Compiler assumes we want an exact copy of each member variable… is that really what we want here?



What happens when a class manages dynamic memory?



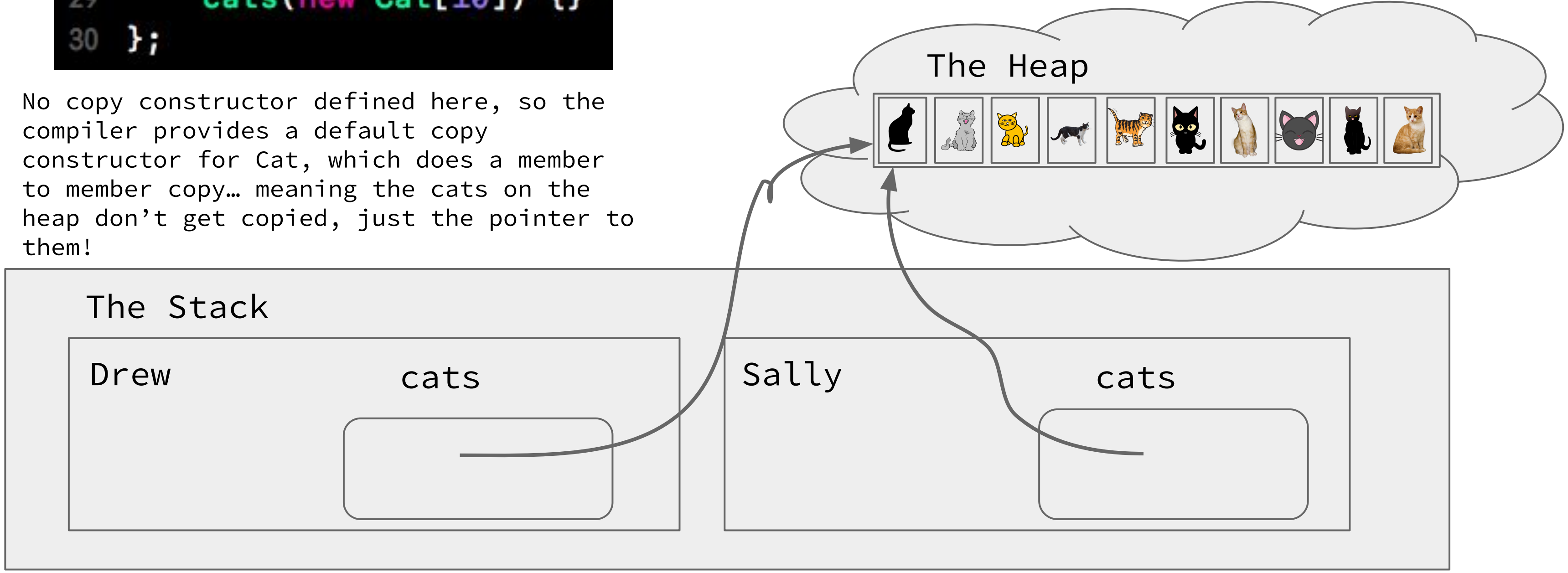
```
class CatLover {
private:
      Cat * cats;
public:
      CatLover() :
      cats(new Cat[10]) {}
};
```



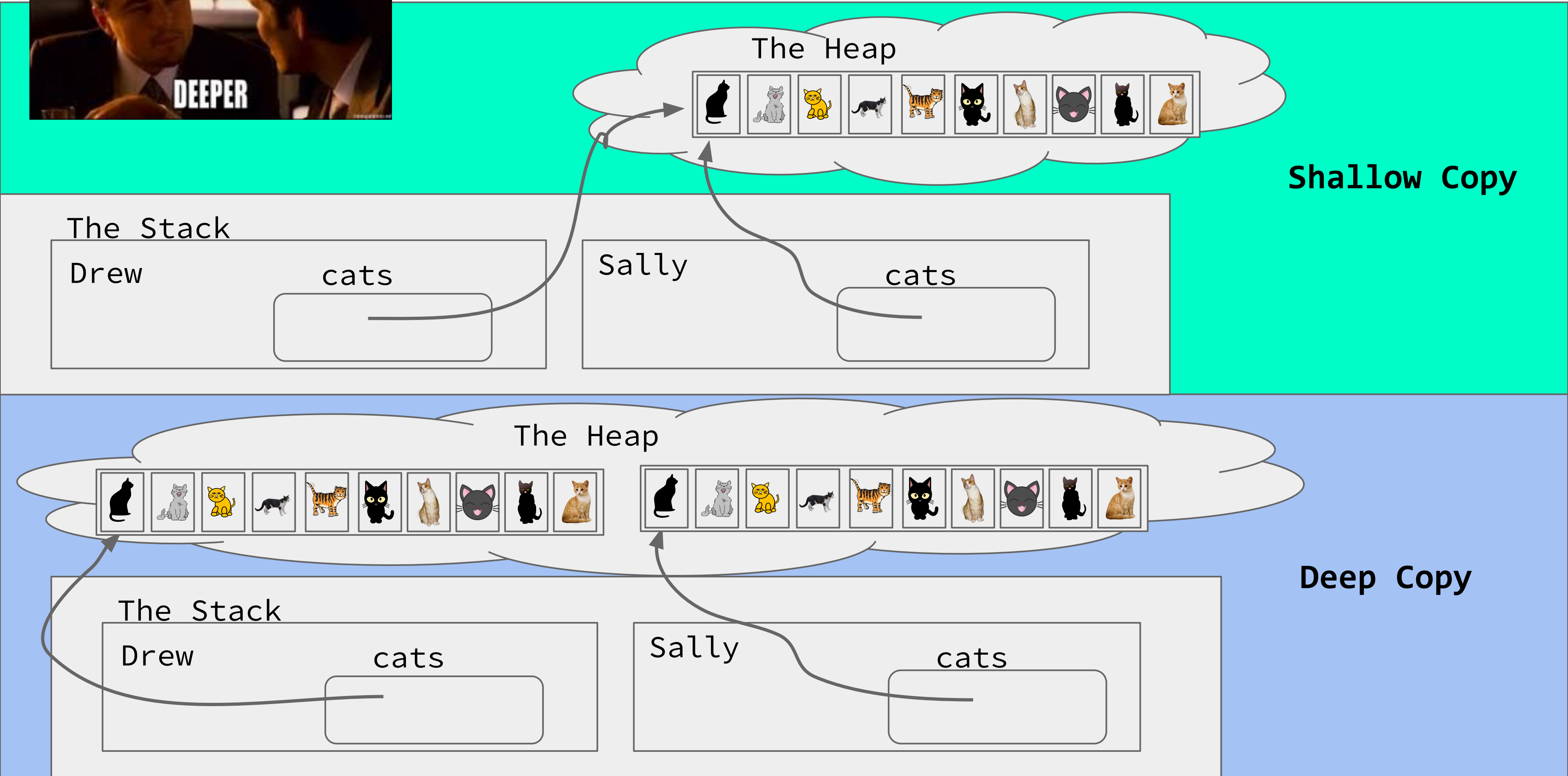
```
CatLover Drew;
CatLover Sally = CatLover(Drew);
```

Sally gets to share Drew's cats!

No copy constructor defined here, so the compiler provides a default copy constructor for Cat, which does a member to member copy… meaning the cats on the heap don't get copied, just the pointer to them!

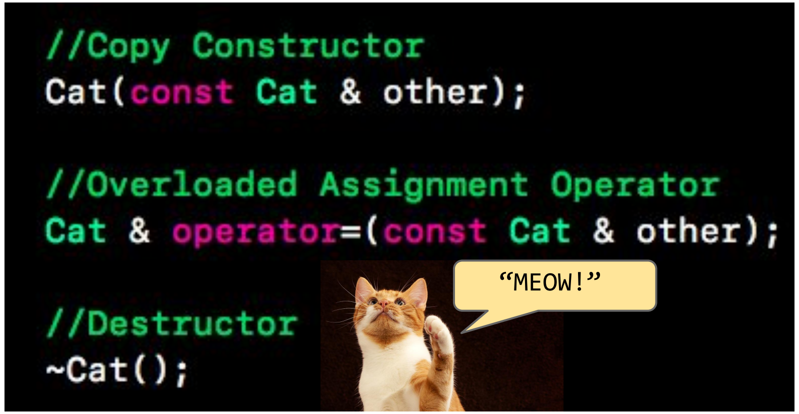


Shallow Copy VS Deep Copy

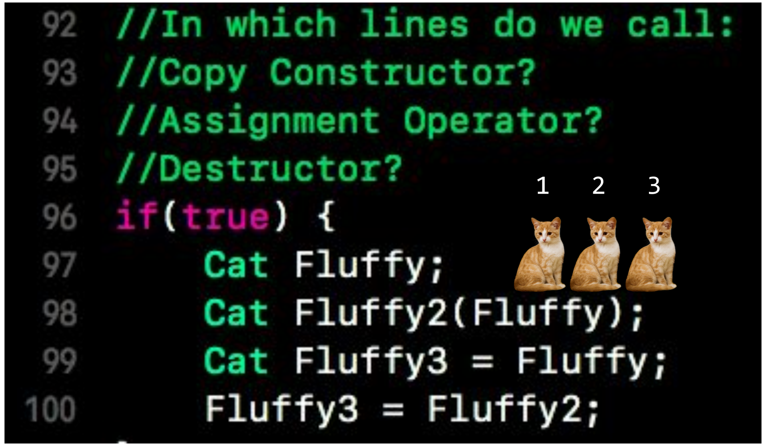


The Big Three

- Copy Constructor
    - Initializes an object by making it a “copy” of another object
- Assignment Operator
    - Assigns an already initialized object to be equal to another object
- Destructor
    - Gets an object’s affairs in order before it is destroyed
- When does my class need The Big Three?
    - (Probably whenever you say “new”!)



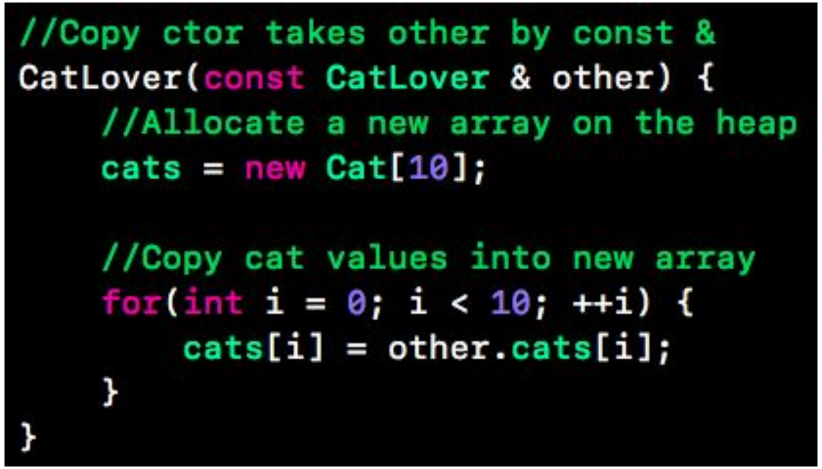
```
//Copy Constructor
Cat(const Cat & other);

//Overloaded Assignment Operator
Cat & operator=(const Cat & other);

//Destructor
~Cat();
```



```
92  //In which lines do we call:
93  //Copy Constructor?
94  //Assignment Operator?
95  //Destructor?
96  if(true) {
97      Cat Fluffy;
98      Cat Fluffy2(Fluffy);
99      Cat Fluffy3 = Fluffy;
100     Fluffy3 = Fluffy2;
```

Writing a Copy Constructor

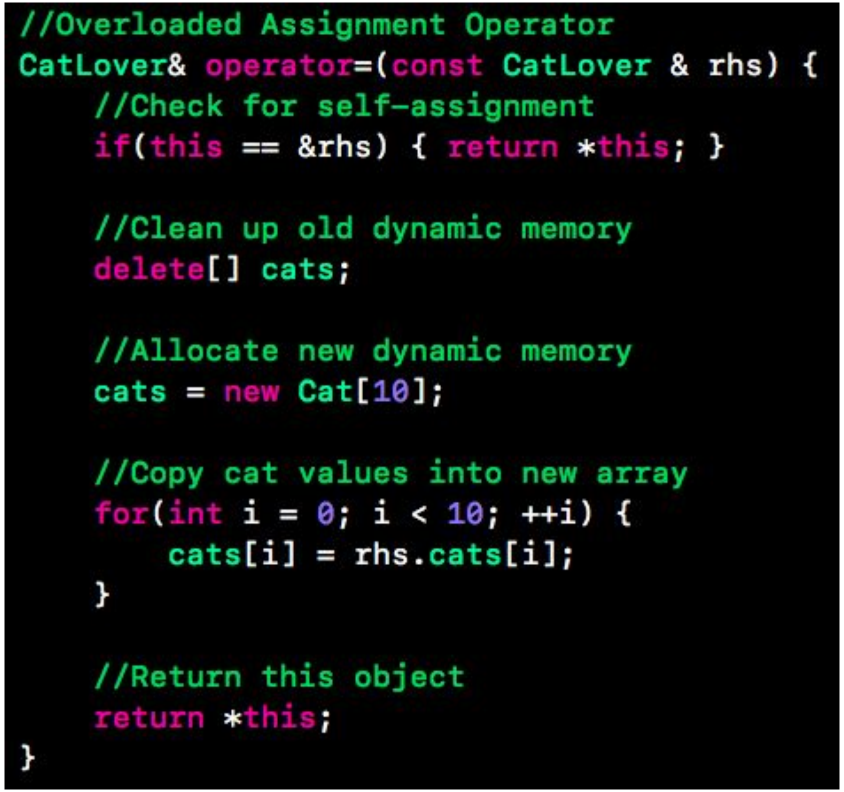
- We need custom copy constructors to copy over objects' dynamic memory
- We will take in the other object by const reference
- We will allocate new dynamic memory and copy over values



```
//Copy ctor takes other by const &
CatLover(const CatLover & other) {
    //Allocate a new array on the heap
    cats = new Cat[10];

    //Copy cat values into new array
    for(int i = 0; i < 10; ++i) {
        cats[i] = other.cats[i];
    }
}
```

Writing an Overloaded Assignment Operator

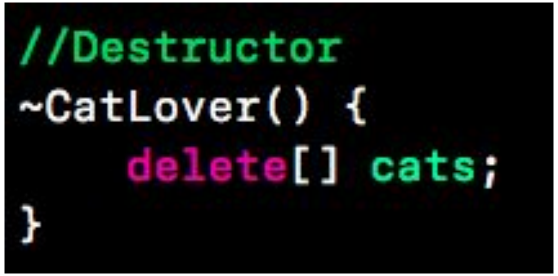
- Think about why copy constructors were necessary - assignment operators are necessary for almost the same reason!
- Called when an already initialized object is assigned
- Key tips:
  - Take parameter by const &
  - Check for self-assignment
  - Take care of old dynamic memory
  - Allocate new dynamic memory and copy values over
  - Return item being assigned into by reference
- Why do we need to check for self-assignment?



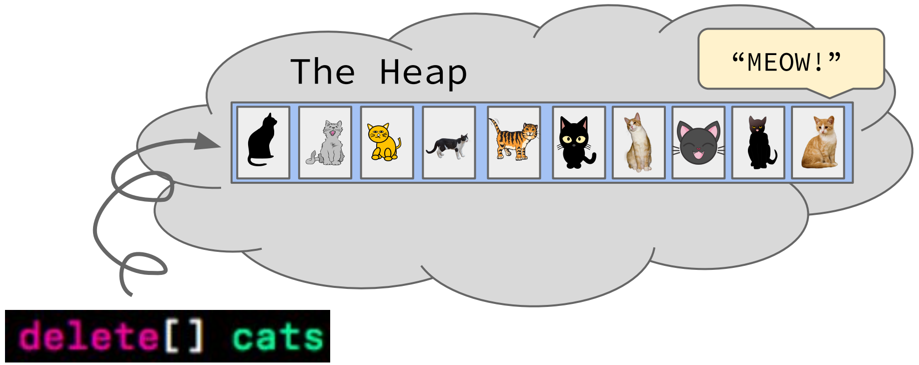
```cpp
//Overloaded Assignment Operator
CatLover& operator=(const CatLover & rhs) {
    //Check for self-assignment
    if(this == &rhs) { return *this; }

    //Clean up old dynamic memory
    delete[] cats;

    //Allocate new dynamic memory
    cats = new Cat[10];

    //Copy cat values into new array
    for(int i = 0; i < 10; ++i) {
        cats[i] = rhs.cats[i];
    }

    //Return this object
    return *this;
}
```

Writing a Destructor

- All objects need to clean up their dynamic memory before they are destroyed - otherwise we will have no way to clean it up and we'll leak a lot of memory.
- Make sure you delete all of the dynamic memory the object manages.
- Note: make your destructors virtual if there are subtypes involved!

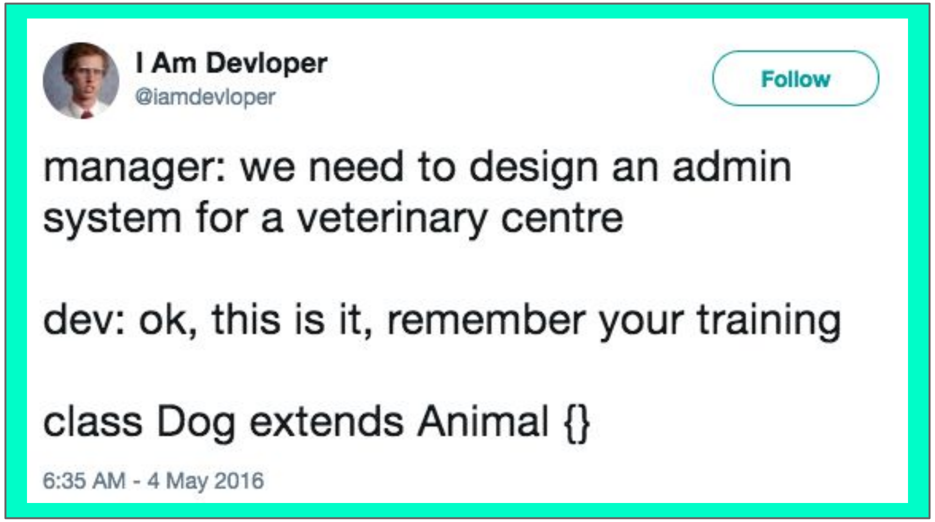


```
//Destructor
~CatLover() {
    delete[] cats;
}
```



```
delete[] cats
```

Worksheet!


I Am Devloper
@iamdevloper

manager: we need to design an admin system for a veterinary centre

dev: ok, this is it, remember your training

class Dog extends Animal {}

6:35 AM - 4 May 2016

Next...

- Please go to:

https://lobster.eecs.umich.edu

- And select from the EECS280 Code portion:

`lab08_BigThree.cpp`

- Step through this with a partner for a few minutes

Moving On!

- Lab 8
- Project 4 Questions?