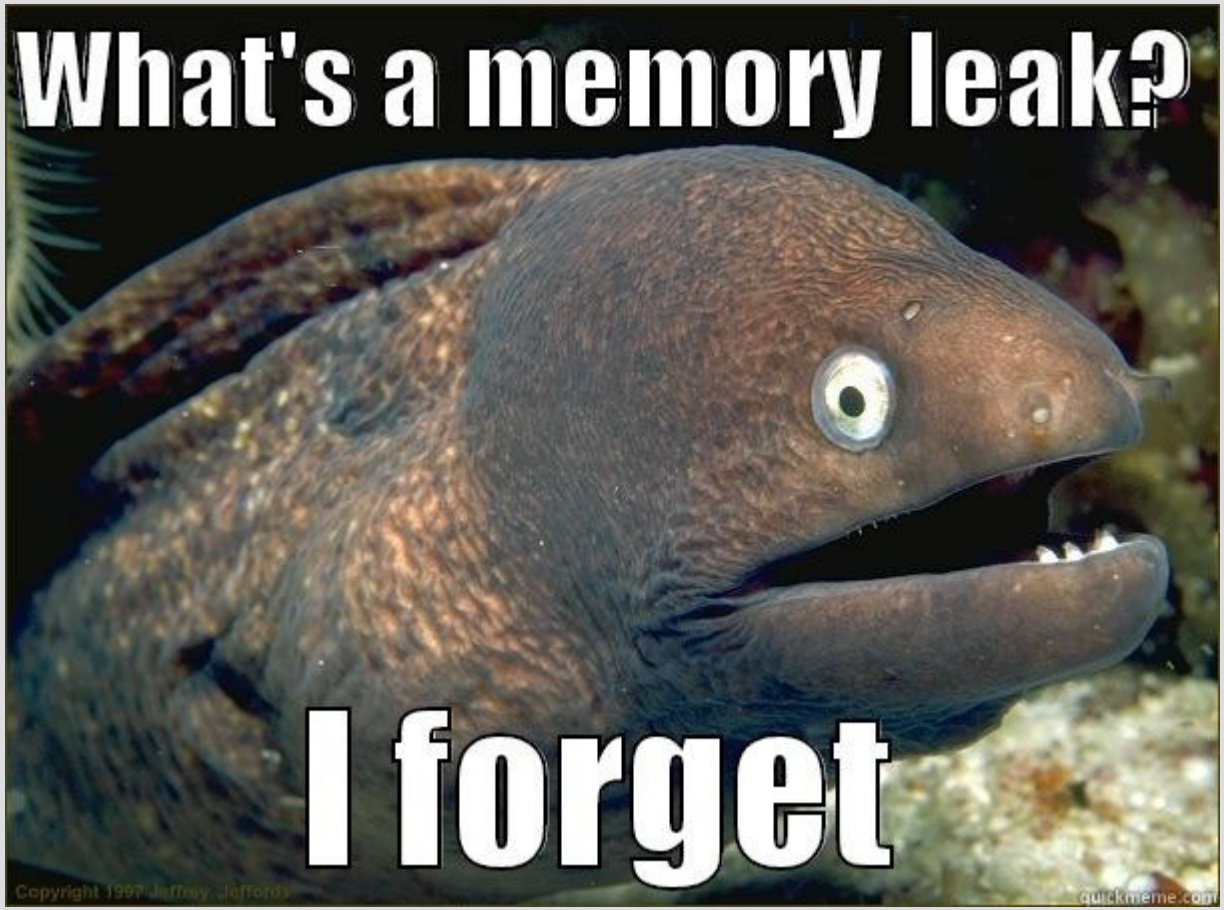
Dynamic memory in class

Ch 9, 11.4, 13.1 & Appendix F



Review: constructors

Constructors are special functions that have the same name as the class

Use a constructor to create an instance of the class (i.e. an object of the blueprint)

```
// all three the same
string a = string("one way");
string b("another way");
string c = "overloaded operator way";
```

Constructors + dynamic

What if we have a variable inside a class that uses dynamic memory?

```
simple::simple()
{
    xArray = new int[3];
}
```

```
class simple{
public:
    int* xArray;
    simple();
};
```

When do we stop using this class?
What do we do if the int* was private?

(See: classMemoryLeak.cpp)

Constructors + dynamic

Often, we might want a class to retain its information until the instance is deleted

This means either:

1. Variable's scope ends (automatically deleted)

```
while(true)
{
    Leaky oops;
}
```

oops out of scope = gone

2. You manually delete a dynamically created class with the delete command

Destructors

Just as a constructor **must** run when a class is created...
A <u>destructor</u> will always run when a class object/instance/variable is deleted

Destructors (like constructors) must have the same name as the class, but with a ~:

```
public:
    Unleaky();
    ~Unleaky();
```

constructor

destructor

(See: classMemoryLeakFixed.cpp)

Destructors

A good analogy is file I/O, as there are 3 steps:

1. Open the file (read or write)
2. Use the file
3. Close the file

The constructor is basically requiring step 1 to happen

Do you want #3 to be automatic or explicit?

Destructors

The benefit of destructors is the computer will run them for you when a variable ends

This means you do not need to explicitly tell it when to delete the dynamic memory, simply how it should be done

This fits better with classes as a blueprint that is used in other parts of the program (see: destructor.cpp)