CS 32 Bootcamp

03—Data Abstraction, C++ Classes

Classes and Objects

- What are objects?
- What are primitives?
- What are classes?

```
class Pizza {
public:
    Pizza(int slices) {
        this->slices = slices;
    }
    void bake() {
        cout << "Smells amazing";
    }
private:
    vector<string> toppings;
    int slices;
};
```

Constructors

- Space is created for the object
- Each of the object's non-primitive members is **default-constructed** in order
- The object's constructor is called (if it is defined)

Challenge: what does this print?

(see pizza.cpp)

Calling a constructor

To initialize a variable:

```
Pizza my_pizza(4);
```

To create a temporary object:

```
eat(Pizza(4))
```

(`eat` is a function that takes a `Pizza`)

There are many other ways to initialize objects in C++…

Default constructors

- A default constructor is a constructor with no arguments
- E.g. `Sauce()` and `Crust()` in the challenge example

Initializer lists

- Recall this statement:
  Each of the object's non-primitive members is **default-constructed** in order
- What if the non-primitive member doesn't have a default constructor?
  - E.g. what if we had to specify a name for the sauce?

Initializer lists

Initializer list items are called when constructing class members (before the constructor body runs)

Syntax:

```
MyClass(constructor params) : member1(ctor args),
                              member2(ctor args),
                              member3(ctor args) {
    constructor body
}
```

Destructors

What happens when an object is destroyed

- The object's own destructor is called first
- Members destructors are called in reverse order
- The memory for the object is freed

This is the *reverse* of object initialization!

Destructors

Syntax:

```
~MyClass() {  // Cannot take params
    Destructor body
}
```