Destructors

An object will often reserve memory slots from the operating system while it runs

A destructor guarantees that reserved memory is freed when an object goes away

```
~Gassy()
{
    //destructor code
}
```

- Must NOT have any parameters
- Must NOT return a value
- Implicit default destructor: C++ will define a destructor that ensures objects properly go away when they go out of scope

Scenarios that need destructor: any time a class allocates a system resource

- Reserve memory with `new`
  - Free allocated memory with `delete`
- Open a disk file
  - Close the disk file
- Connect to another computer over the network
  - Disconnect from other computer

Order of Destruction

- Object goes out of scope -> destructor called

  ```
  int main()
  {
     HungryNerd carey;
  } <- carey's destructor is called
  ```

  - Carey's destructor (outer destructor ran first)

    ```
    ~HungryNerd()
    {
        myBelly.eat(); //last meal
    ```

```
        myBrain.think(); //last thought
    }
```

- Call destructor of member variables in reverse order of construction

  ```
  ~HungryNerd()
  {
      myBelly.eat(); //last meal
      myBrain.think(); //last thought
      //Member variables can still be used here
  }
  Call myBrain's destructor
  Call myBelly's destructor
  ```

- Steps of destruction:

  1. Execute the body of the destructor
  2. Destory each data member:
     - If built-in type, do nothing -If class type, call that class's destructor for that member
  3. ...

When is a destructor called

- Local objects destructed when they go out of scope
- Local variables defined in a block destructed when the block finishes
- Dynamically allocated variables destructued ONLY when `delete` is called
- Called *N* times when you define an array of *N* items at the end of scope

Resource Management