UCLA

CS 32 Data Structures

Week 2, Section B

Jack Gong

UCLA

Course Website:

http://web.cs.ucla.edu/classes/winter19/cs32/

Office Hours:

Tuesdays 12:30PM – 3:30PM, Boelter Hall 3256S

cgpy7@g.ucla.edu

Discussions:

Fridays 10:00AM – 11:50AM, Boelter 5436

Midterm Exams:

Wednesday, January 30, 5PM

Tuesday, February 26, 5PM

Final Exam:

Saturday, March 16, 11:30AM

UCLA

Office Hours:

Wednesdays 9:30AM - 11:30AM, Boelter Hall 3256S

Other LAs' office hours posted on the class website

LA Worksheets:

Optional practice problems (on the website too!)

LA Workshop: Technical Interview Skills

Next Tuesday (1/22), 11am-12pm, Eng VI 289

Excited to meet you guys :)

UCLA

Constructors are called whenever an object of the class is **created**.

If the class contains objects from other class as a member variable, construct the member variable **first**.

Destructor are called whenever the object gets **destroyed**.

If the class contains objects from other class as a member variable, destruct the member variable **last**.

There are two ways to create and destroy the object:

```
//Class A {}...
A object_of_A;
A* pointer_to_A_object = new A();
```

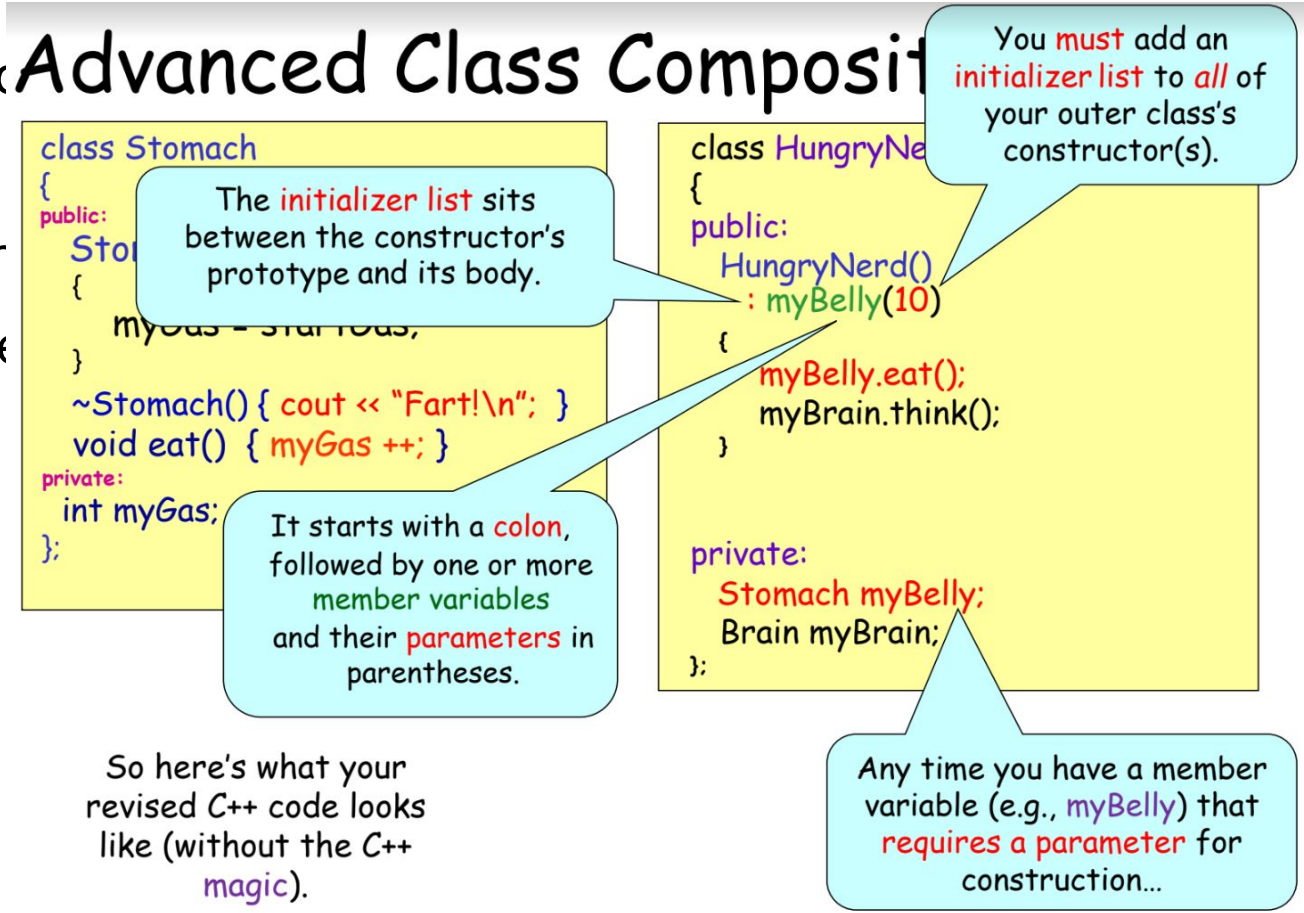
https://repl.it/@BruinUCLA/Order-of-construction

UCLA

When c

- Wh and

nee



So here's what your revised C++ code looks like (without the C++ magic).

UCLA

More constructor stuff in c++:

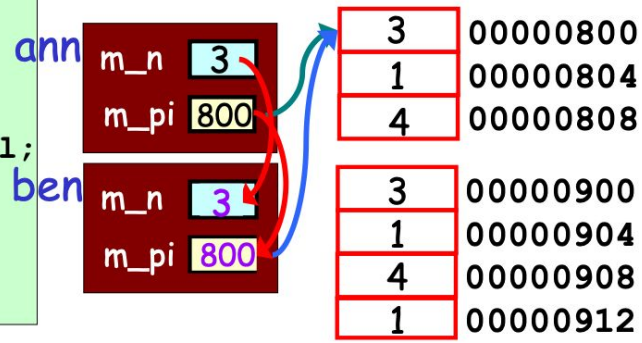
- Default constructor: Empty constructor, does nothing.
- Copy constructor: Initializes an object using another object of the same class. (What's the syntax?)
- https://repl.it/@BruinUCLA/Copy-Constructor-Example?language=cpp11&folderId=

UCLA

The Assignment Operator

```
class PiNerd
{
public:
  PiNerd(int n) {
    m_n = n;
    m_pi = new int[n];
    for (int j=0;j<n;j++)
      m_pi[j] = getPiDigit(j);
  }

  ~PiNerd(){delete []m_pi;}

  void showOff()
  {
    for (int j=0;j<n;j++)
      cout << m_pi[j] << endl;
   }
private:
   int *m_pi, m_n;
};
```

```
int main()
{
    PiNerd  ann(3);

    PiNerd  ben(4);

    ben = ann;
}
```



UCLA

By default, c++ does a **shallow** copy

When you work with classes that contains **new** member variables, you have to overload the operator= with a **deep copy** of their member variables.

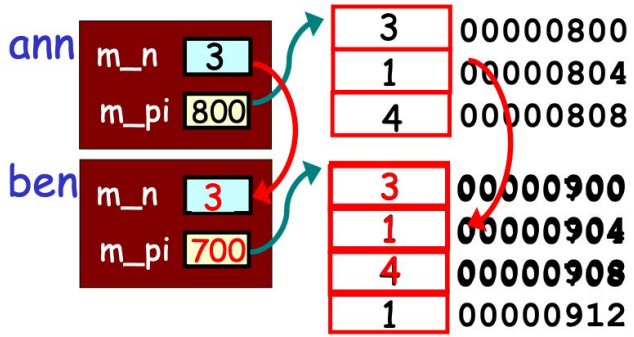
UCLA

Assignment Operator

For such classes, you must define your own *asignment operator!*

Here's how it works for
ben = ann;

1. Free any memory currently held by the target variable (ben).
2. Determine how much memory is used by the source variable (ann).
3. Allocate the same amount of memory in the target variable.
4. Copy the contents of the source variable to the target variable.
5. Return a reference to the target variable.



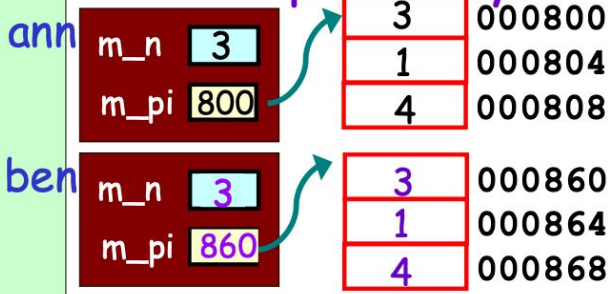
UCLA

The Assignment Operator

```
class PiNerd
{
public:
  PiNerd(int n) { … }
  ~PiNerd(){ delete[]m_pi; }

  // assignment operator:
  PiNerd &operator=(const PiNerd &src)
  {
    delete [] m_pi;
    m_n = src.m_n;
    m_pi = new int[m_n];
    for (int j=0;j<m_n;j++)
       m_pi[j] = src.m_pi[j];
    return *this;
  }
  void showOff() { ... }

private:
   int *m_pi, m_n;
};
```

```
int main()
{
  PiNerd ann(3);
  PiNerd ben(4);

  ben = ann;

}// ann's d'tor called, then ben's
```

… and everything is freed perfectly!

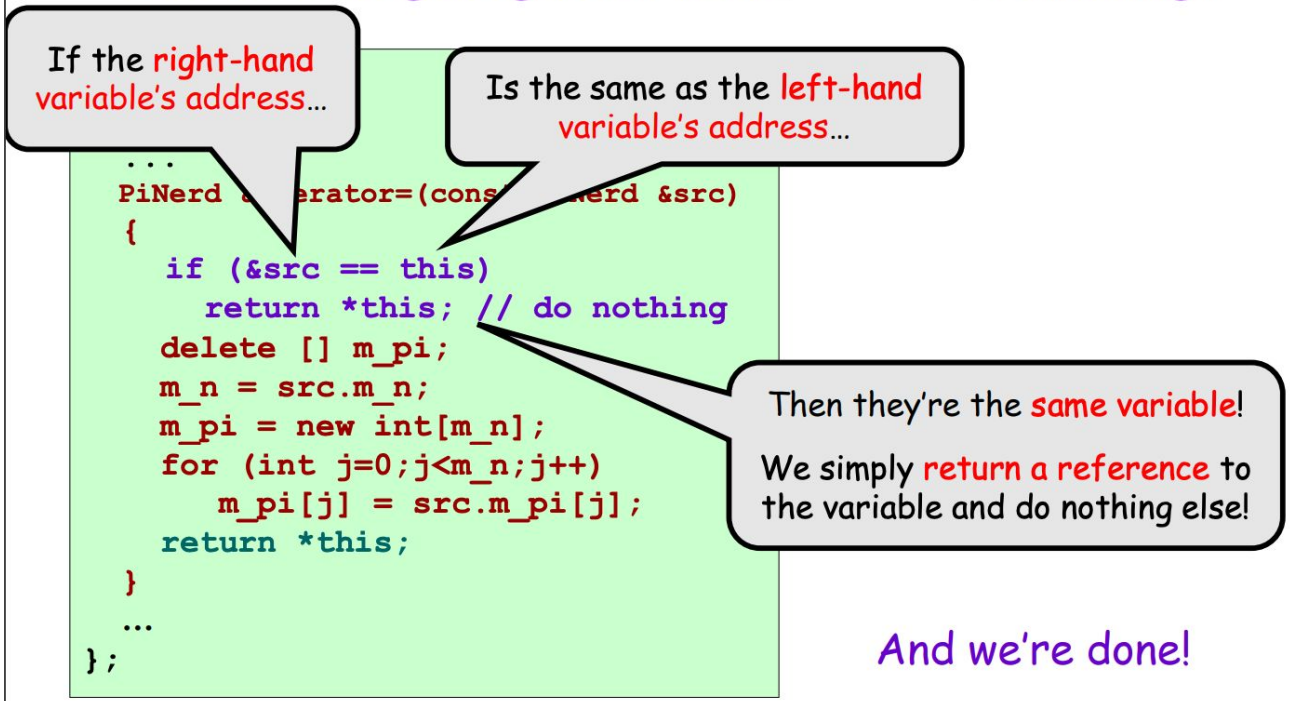


UCLA

The Assignment Operator

The fix:

Our assignment operator function must check to see if a variable is being assigned to itself, and if so, do nothing…



And we're done!

UCLA

Questions?