CS 225
Data Structures

January 28 - Lifecycle
Wade Fagen-Ulmschneider, Craig Zilles
Copy Constructor
Copy Constructor

Automatic Copy Constructor

Custom Copy Constructor
#pragma once

namespace cs225 {

class Cube {

    public:
    Cube();
    Cube(double length);

    double getVolume() const;
    double getSurfaceArea() const;

    private:
    double length_;  
};

}

namespace cs225 {

    Cube::Cube() {
        length_ = 1;
        cout << "Default ctor" << endl;
    }

    Cube::Cube(double length) {
        length_ = length;
        cout << "1-arg ctor" << endl;
    }

    ... // ...

/*
 * Creates a new Cube that contains the exact volume
 * of the volume of the two input Cubes.
 */

Cube joinCubes(Cube c1, Cube c2) {
    double totalVolume = c1.getVolume() + c2.getVolume();
    double newLength = std::pow( totalVolume, 1.0/3.0 );
    Cube result(newLength);
    return result;
}

int main() {
    Cube *c1 = new Cube(4);
    Cube *c2 = new Cube(5);
    Cube c3 = joinCubes(*c1, *c2);
    return 0;
}
## Calls to constructors

<table>
<thead>
<tr>
<th></th>
<th>By Value</th>
<th>By Pointer</th>
<th>By Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>void foo(Cube a) { ... }</td>
<td>void foo(Cube *a) { ... }</td>
<td>void foo(Cube &amp;a) { ... }</td>
</tr>
<tr>
<td>Cube::Cube()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cube::Cube(double)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cube::Cube(const Cube&amp;)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
/ * Creates a new Cube that contains the exact volume * of the volume of the two input Cubes. */
Cube joinCubes(Cube * c1, Cube * c2) {
    double totalVolume = c1->getVolume() + c2->getVolume();
    double newLength = std::pow( totalVolume, 1.0/3.0 );
    Cube result(newLength);
    return result;
}

int main() {
    Cube *c1 = new Cube(4);
    Cube *c2 = new Cube(5);
    Cube c3 = joinCubes(c1, c2);
    return 0;
}
/ * Creates a new Cube that contains the exact volume * of the volume of the two input Cubes. */ Cube joinCubes(Cube & c1, Cube & c2) { double totalVolume = c1.getVolume() + c2.getVolume(); double newLength = std::pow( totalVolume, 1.0/3.0 ); Cube result(newLength); return result; } int main() { Cube *c1 = new Cube(4); Cube *c2 = new Cube(5); Cube c3 = joinCubes(*c1, *c2); return 0; }
Upcoming: Theory Exam #1

Theory Exam #1

• Starts this Thursday

• 70 points
• 14 MC, 1 code-reading

• Topic List: posted to web page soon
Wade Monday
Honors Section

CS 225 offers a one-credit add on honors section!

What is data science?

- Algorithms
- Python
- Data Structures
- pandas
- Visualizations
- JavaScript
- d3.js
Honors Section

Course Starts: Thursday, February 14, 2019
Meets: Thursdays: 5:00 – 5:50pm, 1404 Siebel Center

Taught By: Wade Fagen-Ulmschneider (CS faculty)

Open to EVERYONE – not required to be part of an honors program. Fulfills HCLA, James Scholar, etc.

CS 296, Section 25 (CRN: 31262)
MP1 Deadline

Programming is hard!
MP1 Deadline

Programming is hard!
Every MP in CS 225 will have an automatic 24-hour grace period after the due date.

Due: Monday, 11:59pm
Grade Period until: Tuesday, 11:59pm
MP1 Deadline

**Programming is hard!**
Every MP in CS 225 will have an automatic 24-hour grace period after the due date.

- **Due:** Monday, 11:59pm
- **Grade Period until:** Tuesday, 11:59pm

Since the MP will past-due, **there are absolutely no office/lab hours on Tuesdays.**
Registration

The last chance to register for CS 225 is today. We will not being doing any late adds.

If you’ve registered late, everything so far is due this Tuesday, January 29th @ 11:59pm.

- lab_intro
- lab_debug
- mp1
#pragma once

#include "cs225/Cube.h"

using cs225::Cube;

class Tower {
public:
  Tower(Cube c, Cube *ptr, const Cube &ref);
  Tower(const Tower &other);

private:
  Cube cube_;  // private data member
  Cube *ptr_;  // private data member
  const Cube &ref_;  // private data member
};
10  Tower::Tower(const Tower & other) {
11    cube_ = other.cube_;  
12    ptr_ = other.ptr_;  
13    ref_ = other.ref_;  
14  }
Tower::Tower(const Tower & other) {
    cube_ = other.cube_
    ptr_ = other.ptr_
    ref_ = other.ref_
}
Tower::Tower(const Tower & other) {
    cube_ = other.cube_;  
    ptr_ = other.ptr_;    
    ref_ = other.ref_;    
}
Tower::Tower(const Tower & other) {
    // Deep copy cube_
    
    // Deep copy ptr_
    
    // Deep copy ref_
}
Destructor

[Purpose]:

Destructor

[Purpose]: **Free any resources maintained by the class.**

**Automatic Destructor:**
1. Exists only when no custom destructor is defined.

2. [Functionality]:

[Invoked]:

#pragma once

namespace cs225 {

class Cube {

public:

    Cube();
    Cube(double length);
    Cube(const Cube & other);
    ~Cube();

    double getVolume() const;
    double getSurfaceArea() const;

private:

    double length_;  

};
}

namespace cs225 {

Cube::Cube() {
    length_ = 1;
    cout << "Default ctor"  
    << endl;
}

Cube::Cube(double length) {
    length_ = length;
    cout << "1-arg ctor"  
    << endl;
}
... // ...