CS 225
Data Structures

September 10 - Lifecycle
Wade Fagen-Ulmschneider
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>Cube::Cube()</th>
<th>By Value</th>
<th>By Pointer</th>
<th>By Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>void foo(Cube a) { ... }</td>
<td>void foo(Cube *a) { ... }</td>
<td>void foo(Cube &amp;a) { ... }</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;
}
Mattox Monday
Upcoming: Theory Exam #1

Theory Exam #1
• Starts this Thursday

• Topic List:

https://courses.engr.illinois.edu/cs225/fa2018/exams/exam-theory1/
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, Sept. 11 @ 11:59pm.

• lab_intro
• lab_debug
• mp1
```cpp
#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_;    // Define private member variables here
    Cube *ptr_;    // Define private member variables here
    const Cube &ref;
};
```
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_;  
}

waf@tiebl-2215-02:/mnt/c/Users/waf/Desktop/cs225/_lecture/06-lifecycle$ make
clang++ -std=c++1y -stdlib=libc++ -O0 -Wall -Wextra -pedantic -lpthread -lm main.cpp cs225/Cube.cpp Tower.cpp -o main
Tower.cpp:10:8: error: constructor for 'Tower' must explicitly initialize the reference member 'ref_'
  Tower::Tower(const Tower & other) {  
           ^
./Tower.h:14:17: note: declared here
   const Cube &ref_;  
          ^
Tower.cpp:20:8: error: no viable overloaded '='
    ref_ = other.ref_;  
    ^

Note: declared here
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
#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;
    }

    // ...
    ... // ...
}
Operators that can be overloaded in C++

<table>
<thead>
<tr>
<th>Category</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>+ - * / % ++ --</td>
</tr>
<tr>
<td>Bitwise</td>
<td>&amp;</td>
</tr>
<tr>
<td>Assignment</td>
<td>=</td>
</tr>
<tr>
<td>Comparison</td>
<td>== != &gt; &lt; &gt;= &lt;=</td>
</tr>
<tr>
<td>Logical</td>
<td>! &amp;&amp;</td>
</tr>
<tr>
<td>Other</td>
<td>[ ] () -&gt;</td>
</tr>
</tbody>
</table>
```
#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_;  
  };  
};
```