int *x;
int size = 3;

x = new int[size];

for (int i = 0; i < size; i++) {
    x[i] = i + 3;
}

delete[] x;
/*
 * 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;
}
/ * 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;
}
## Parameter Passing Properties

<table>
<thead>
<tr>
<th></th>
<th>By Value</th>
<th>By Pointer</th>
<th>By Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Exactly what is copied when</strong></td>
<td>void foo(Cube a) { ... }</td>
<td>void foo(Cube *a) { ... }</td>
<td>void foo(Cube &amp;a) { ... }</td>
</tr>
<tr>
<td><strong>the function is invoked?</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Does modification of the</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>passed in object modify</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>the caller’s object?</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Is there always a valid</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>object passed in to the</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>function?</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Speed</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Programming Safety</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MP1

**Due:** Monday, Sept. 10\textsuperscript{th} (11:59pm)

**Share your art work:**

- On our piazza, in the “MP1 - Artwork Sharing” thread
- On social media:
  - If your post is **public** and contains **#cs225**, I’ll throw it a like/heart and so will some of your peers! 😊

**My promise:** I will look at all the artwork after the submission deadline. Course staff and I will give +1 to all that stand out!
Using `const` in function parameters
/*
 * Creates a new Cube that contains the exact volume
 * of the volume of the two input Cubes.
 */

Cube joinCubes(const Cube c1, const 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(const Cube * c1, const 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(const Cube & c1, const 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;
}
make clang++ -std=c++1y -stdlib=libc++ -O0 -Wall -Wextra -pedantic -lpthread -lm joinCubes-byValue-const.cpp cs225/Cube.cpp -lm -o joinCubes-byValue-const

joinCubes-byValue-const.cpp:16:24: error: member function 'getVolume' not viable: 'this' argument has type 'const cs225::Cube', but function is not marked const
double totalVolume = c1.getVolume() + c2.getVolume();
       ^
./cs225/Cube.h:9:14: note: 'getVolume' declared here
double getVolume();
         ^

joinCubes-byValue-const.cpp:16:41: error: member function 'getVolume' not viable: 'this' argument has type 'const cs225::Cube', but function is not marked const
double totalVolume = c1.getVolume() + c2.getVolume();
       ^
./cs225/Cube.h:9:14: note: 'getVolume' declared here
double getVolume();
         ^

2 errors generated.
Makefile:19: recipe for target 'joinCubes-byValue-const' failed
make: *** [joinCubes-byValue-const] Error 1
const as part of a member functions’ declaration
#pragma once

namespace cs225 {
    class Cube {
        public:
            Cube();
            Cube(double length);
            double getVolume() const;
            double getSurfaceArea() const;
        
        private:
            double length_;  
    };
}

#include "Cube.h"

namespace cs225 {
    Cube::Cube() {
        length_ = 1;
    }

    Cube::Cube(double length) {
        length_ = length;
    }

    double Cube::getVolume() const {
        return length_ * length_ * length_;  
    }

    double Cube::getSurfaceArea() const {
        return 6 * length_ * length_;  
    }
}
/*
 * Creates a new Cube that contains the exact volume
 * of the volume of the two input Cubes.
 */
Cube joinCubes(const Cube c1, const 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;
}
Copy Constructor

[Purpose]:
All copy constructors will
Copy Constructor

Automatic Copy Constructor

Custom Copy Constructor
#pragma once

namespace cs225 {
    class Cube {
        public:
            Cube();
            Cube(double length);
            Cube(const Cube & other);
            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(const Cube c1, const 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(const Cube * c1, const 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(const Cube & c1, const 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;
}
MORE MONEY, MORE TIME!

Deadlines EXTENDED for the following scholarships:

- **September 12**  John Deere WCS Scholarship ($2,000 x 4!)
- **September 12**  JPMorgan Chase WCS Scholarship ($2,500 x 2!)
- **September 12**  NEW! IMC Trading Scholarship ($7,500 x 2!)

Find details at: [http://go.cs.illinois.edu/AwardDeadlines](http://go.cs.illinois.edu/AwardDeadlines)

Questions? Contact Samantha at shendon@illinois.edu