/*
 * Creates a new sphere that contains the exact volume
 * of the volume of the two input spheres.
 */

Sphere joinSpheres(const Sphere &s1, const Sphere &s2) {
    double totalVolume = s1.getVolume() + s2.getVolume();
    double newRadius = std::pow((3.0 * totalVolume) / (4.0 * 3.141592654), 1.0/3.0);
    return result(newRadius);
}

int main() {
    Sphere *s1 = new Sphere(4);
    Sphere *s2 = new Sphere(5);
    Sphere s3 = joinSpheres(*s1, *s2);
    delete s1; s1 = NULL;
    delete s2; s2 = NULL;
    return 0;
}
/*
  * Creates a new sphere that contains the exact volume
  * of the volume of the two input spheres.
  */
Sphere & joinSpheres(const Sphere &s1, const Sphere &s2) {
    double totalVolume = s1.getVolume() + s2.getVolume();
    double newRadius = std::pow((3.0 * totalVolume) / (4.0 * 3.141592654), 1.0/3.0);
    Sphere result(newRadius);
    return result;
}

int main() {
    Sphere *s1 = new Sphere(4);
    Sphere *s2 = new Sphere(5);
    Sphere s3 = joinSpheres(*s1, *s2);
    delete s1; s1 = NULL;
    delete s2; s2 = NULL;
    return 0;
}
/*
 * Creates a new sphere that contains the exact volume
 * of the volume of the two input spheres.
 */
Sphere & joinSpheres(const Sphere &s1, const Sphere &s2) {
    double totalVolume = s1.getVolume() + s2.getVolume();
    double newRadius = std::pow((3.0 * totalVolume) / (4.0 * 3.141592654), 1.0/3.0);
    Sphere *result = new Sphere(newRadius);
    return *result;
}

int main() {
    Sphere *s1 = new Sphere(4);
    Sphere *s2 = new Sphere(5);
    Sphere s3 = joinSpheres(*s1, *s2);
    delete s1; s1 = NULL;
    delete s2; s2 = NULL;
    return 0;
}
/ * Creates a new sphere that contains the exact volume * of the volume of the two input spheres. */
Sphere *joinSpheres(const Sphere &s1, const Sphere &s2) {
double totalVolume = s1.getVolume() + s2.getVolume();

double newRadius = std::pow((3.0 * totalVolume) / (4.0 * 3.141592654), 1.0/3.0);

return new Sphere(newRadius);
}

int main() {
    Sphere *s1 = new Sphere(4);
    Sphere *s2 = new Sphere(5);
    Sphere *s3 = joinSpheres(*s1, *s2);
    delete s1; s1 = NULL;
    delete s2; s2 = NULL;
    return 0;
}
Upcoming: Theory Exam #1

Theory Exam #1
• Starts tomorrow

• Topic List:
  https://courses.engr.illinois.edu/cs225/sp2018/exams/exam-theory1/

• Review Session:
  Monday, 7:00pm, 1404 Siebel Center

Topics Covered
Topics from lecture:
• Classes in C++
  • Public members functions
  • Private helper functions
  • Private variables
  • Constructors
  • Automatic default constructor
• Namespaces in C++
  • Creating a class that is part of a namespace (eg: Sphere is part of the cs225 namespace)
  • Using a class from a namespace (eg: cs225::Sphere)
  • Purpose and usefulness of namespaces
• Variables
  • Four properties: name, type, location (in memory), and value
  • Primitive vs. user-defined
• Memory
  • Indirection in C++:
    • Reference variables
    • Pointers
    • Differences and trade-offs between each type
    • Stack memory
    • Heap memory
• Functions: Calling and Returning
  • Pass by value, by reference, and by pointer
  • Return by value, by reference, and by pointer

Assignments referenced:
• lab_intro
• lab_debug
• MP1
MP1 Deadline

Programming is hard!
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 be doing any late adds.

If you’ve registered late, everything so far is due this Tuesday, Jan. 30 @ 11:59pm.

• lab_intro
• lab_debug
• mp1
Copy Constructor
Copy Constructor

Automatic Copy Constructor

Custom Copy Constructor
```cpp
#ifndef SPHERE_H
#define SPHERE_H

namespace cs225 {

class Sphere {
 public:
    Sphere();
    Sphere(double r);

double getRadius() const;
    double getVolume() const;

    void setRadius(double r);

 private:
    double r_;  

};
}
#endif

#include "sphere.h"
#include <iostream>

using namespace std;

namespace cs225 {

Sphere::Sphere() : Sphere(1) {
    cout << "Default ctor" << endl;
}

Sphere::Sphere(double r) {
    cout << "1-param ctor" << endl;
    r_ = r;
}
```

... // ...
```
# 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>void foo(Sphere a) { ... }</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>void foo(Sphere *a) { ... }</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>void foo(Sphere &amp;a) { ... }</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sphere::Sphere()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sphere::Sphere(double)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sphere::Sphere(const Sphere&amp;)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
/*
 * Creates a new sphere that contains the exact volume
 * of the volume of the two input spheres.
 */

Sphere joinSpheres(const Sphere s1, const Sphere s2) {
    double totalVolume = s1.getVolume() + s2.getVolume();

    double newRadius = std::pow((3.0 * totalVolume) / (4.0 * 3.141592654), 1.0/3.0);

    Sphere result(newRadius);
    return result;
}

int main() {
    Sphere *s1 = new Sphere(4);
    Sphere *s2 = new Sphere(5);

    Sphere s3 = joinSpheres(*s1, *s2);

    delete s1; s1 = NULL;
    delete s2; s2 = NULL;

    return 0;
}
/*
 * Creates a new sphere that contains the exact volume
 * of the volume of the two input spheres.
 */
Sphere *joinSpheres(const Sphere *s1, const Sphere *s2) {
    double totalVolume = s1->getVolume() + s2->getVolume();
    double newRadius = std::pow((3.0 * totalVolume) / (4.0 * 3.141592654), 1.0/3.0);
    return new Sphere(newRadius);
}

int main() {
    Sphere *s1 = new Sphere(4);
    Sphere *s2 = new Sphere(5);
    Sphere *s3 = joinSpheres(s1, s2);
    delete s1; s1 = NULL;
    delete s2; s2 = NULL;
    return 0;
}
/*
 * Creates a new sphere that contains the exact volume
 * of the volume of the two input spheres.
 */

Sphere & joinSpheres(const Sphere &s1, const Sphere &s2) {
    double totalVolume = s1.getVolume() + s2.getVolume();

    double newRadius = std::pow(
        (3.0 * totalVolume) / (4.0 * 3.141592654),
        1.0/3.0
    );

    // !: Not good practice,
    //    memory not free'd!

    Sphere *result = new Sphere(newRadius);

    return *result;
}

int main() {
    Sphere *s1 = new Sphere(4);
    Sphere *s2 = new Sphere(5);
    Sphere s3 = joinSpheres(*s1, *s2);
    delete s1; s1 = NULL;
    delete s2; s2 = NULL;

    return 0;
}
```cpp
#ifndef UNIVERSE_H_
#define UNIVERSE_H_

#include "Sphere.h"
using namespace cs225;

class Universe {
public:
    Universe(); // default ctor
    Universe(Sphere s, Sphere *q, Sphere &r); // 3-param
    Universe(const Universe &other);
    // ...
private:
    Sphere p_, *q_, &r;
};

#endif
```
Universe::Universe(const Universe & other) {
    p_ = other.p_;    
    q_ = other.q_;  
    r_ = other.r_;  
}
```cpp
10 Universe::Universe(const Universe & other) {
11     p_ = other.p_;  // Error: Constructor for 'Universe' must explicitly initialize the reference member 'r_'
12     q_ = other.q_;
13     r_ = other.r_;  
14 }
```
Universe::Universe(const Universe & other) {
    p_ = other.p_; 
    q_ = other.q_; 
    r_ = other.r_; 
}

Universe.cpp:7:2: error: constructor for 'Universe' must explicitly initialize the reference member 'r_'

Universe::Universe(const Universe & other) :
    p_(other.p_), q_(other.q_), r_(other.r_) {
}
Universe::Universe(const Universe & other) {
    // Deep copy p_
    // Deep copy q_
    // Deep copy r_
}

Destructor
#ifndef SPHERE_H
#define SPHERE_H

namespace cs225 {

class Sphere {

public:
    Sphere();
    Sphere(double r);
    Sphere(const Sphere &s);
    ~Sphere();

private:
    double r_;

};

}  // namespace "cs225"
#endif
<table>
<thead>
<tr>
<th>Operators that can be overloaded in C++</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arithmetic</strong></td>
</tr>
<tr>
<td>+  -  *  /  %  ++  --</td>
</tr>
<tr>
<td><strong>Bitwise</strong></td>
</tr>
<tr>
<td>&amp;</td>
</tr>
<tr>
<td><strong>Assignment</strong></td>
</tr>
<tr>
<td>=</td>
</tr>
<tr>
<td><strong>Comparison</strong></td>
</tr>
<tr>
<td>==  !=  &gt;  &lt;  &gt;=  &lt;=</td>
</tr>
<tr>
<td><strong>Logical</strong></td>
</tr>
<tr>
<td>!  &amp;&amp;</td>
</tr>
<tr>
<td><strong>Other</strong></td>
</tr>
<tr>
<td>[ ]  ()  -&gt;</td>
</tr>
</tbody>
</table>
#ifndef SPHERE_H
#define SPHERE_H

namespace cs225 {
    class Sphere {
    public:
        Sphere();
        Sphere(double r);
        Sphere(const Sphere &s);
        ~Sphere();
    
        // ...

    private:
        double r_;
    }
    ...
}
#endif

#include "sphere.h"

namespace cs225 {
 ...
 // ...
}
CS 225 – Things To Be Doing

Theory Exam 1 starts tomorrow
More Info: https://courses.engr.illinois.edu/cs225/sp2018/exams/

MP1 – Due Tonight (11:59pm)
MP2 Release: Tuesday, Sept 12th – Up to +7 Extra Credit for Early Submission

POTD
Every Monday-Friday – Worth +1 Extra Credit / problem (up to +40 total)