CS 225

Data Structures

Wade Fagen-Ulmschneider

heap-puzzle3.cpp

```cpp
int *x;
int size = 3;

x = new int[size];

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

delete[] x;
```

Upcoming: Theory Exam #1

Theory Exam #1

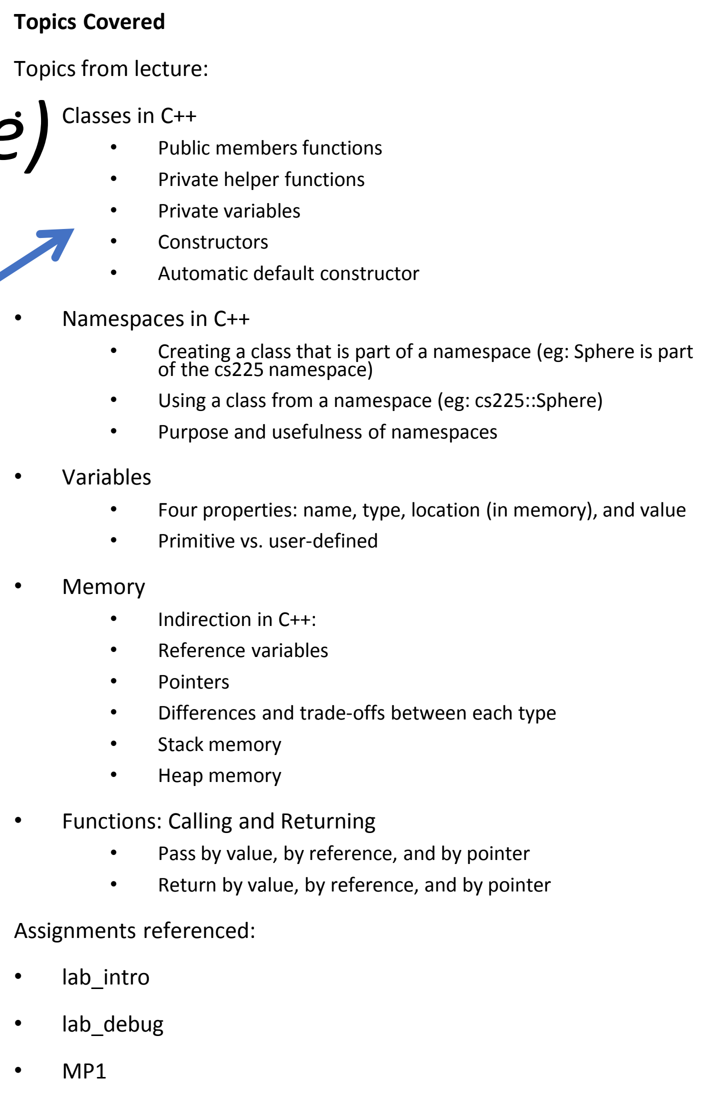
- Starts on Tuesday *(the day after MP1 is due)*

- **Topic List:**

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

- **Review Session:**

  Monday, 7:00pm, 1404 Siebel Center



Topics Covered

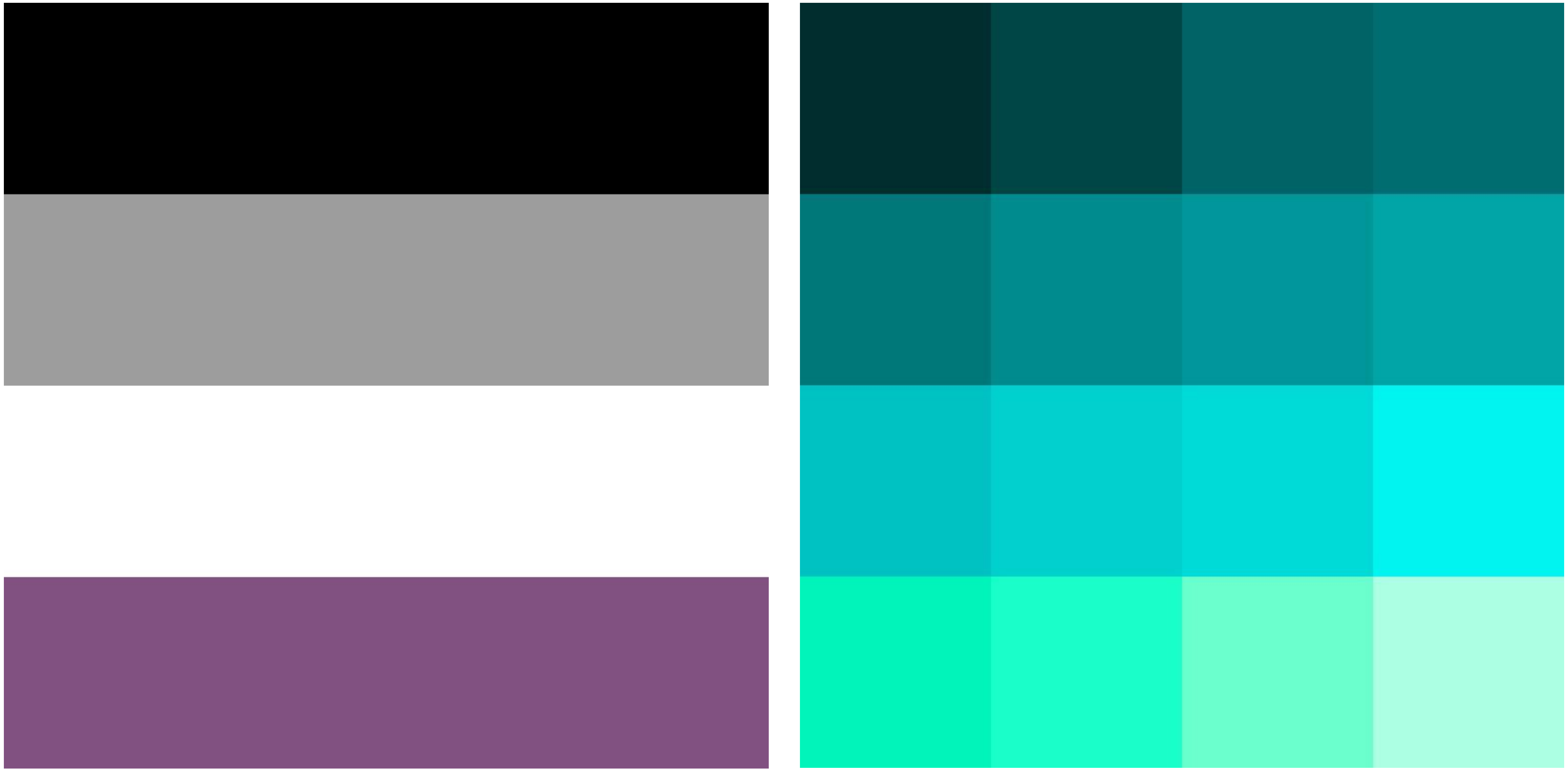
Topics from lecture:

- 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

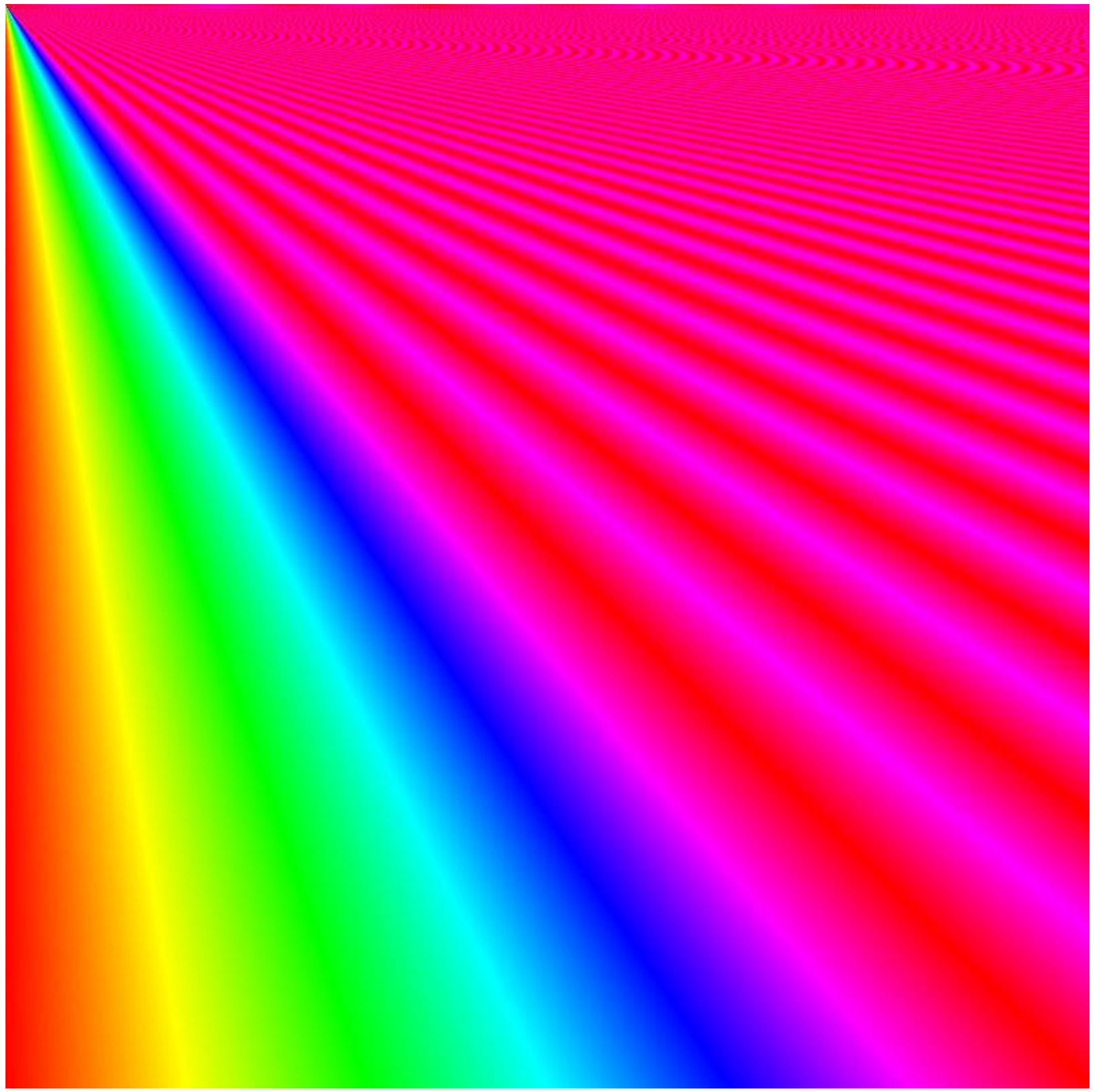
Assignments referenced:

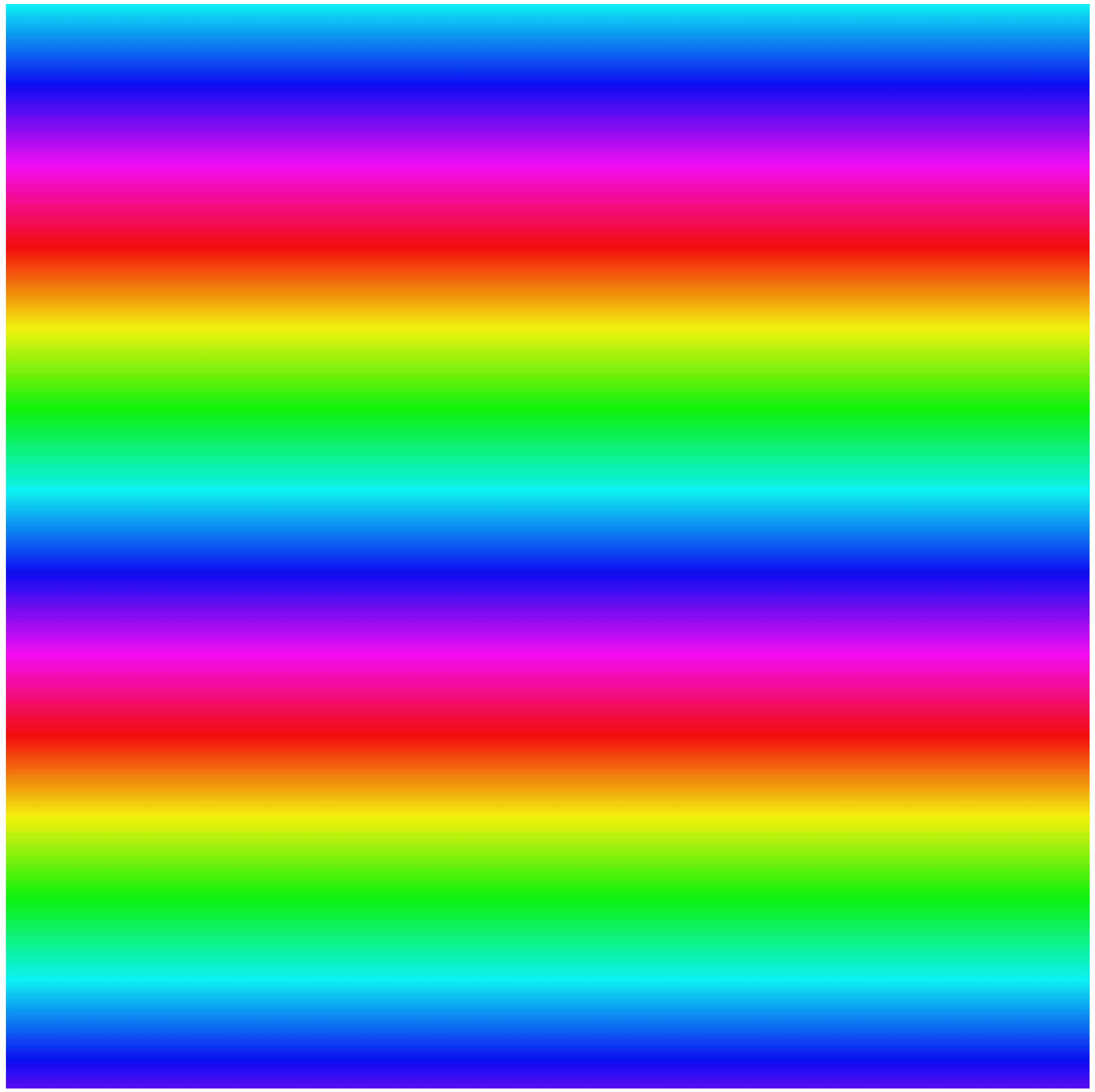
- lab_intro
- lab_debug
- MP1

MP1

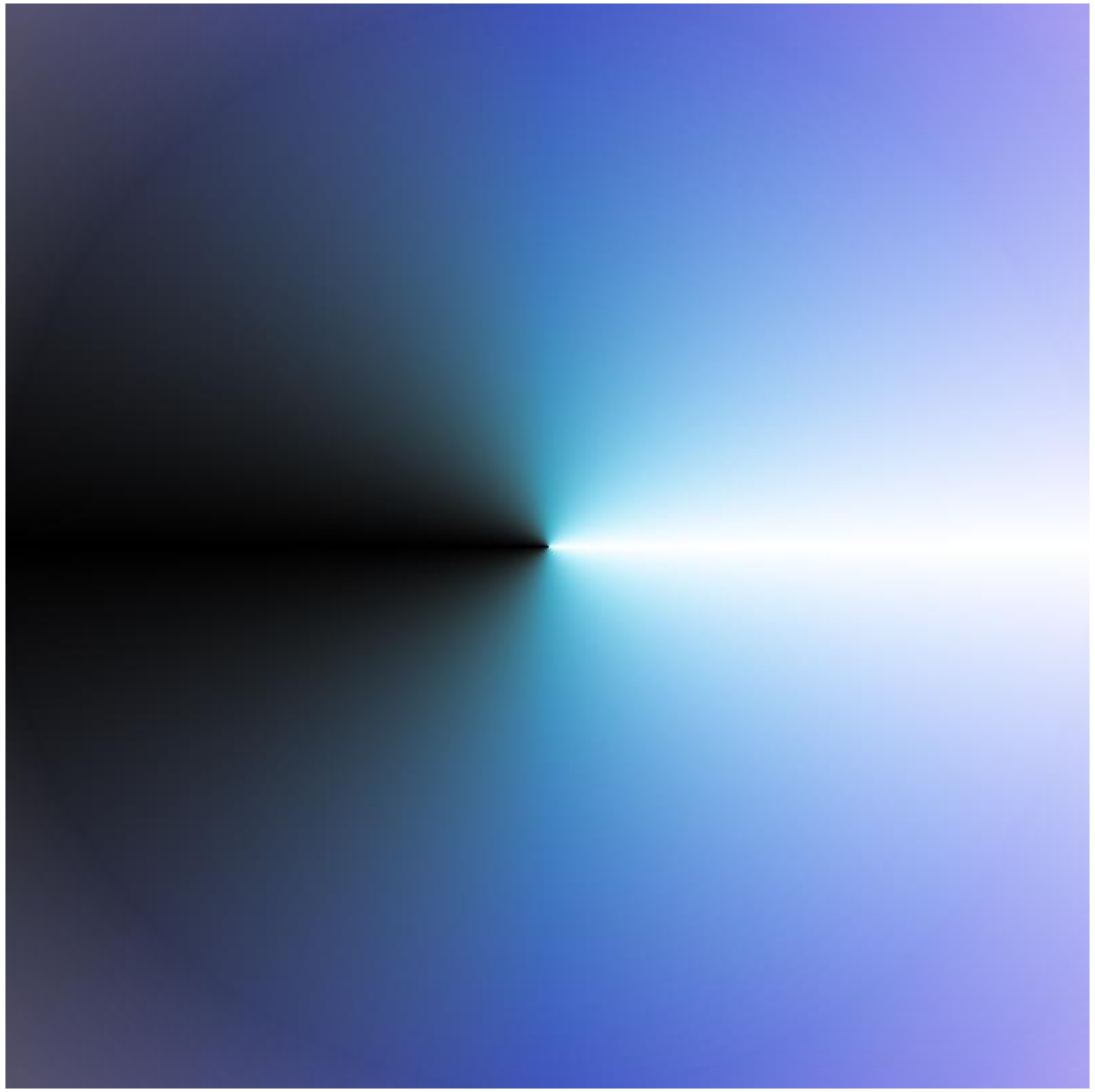


MP1

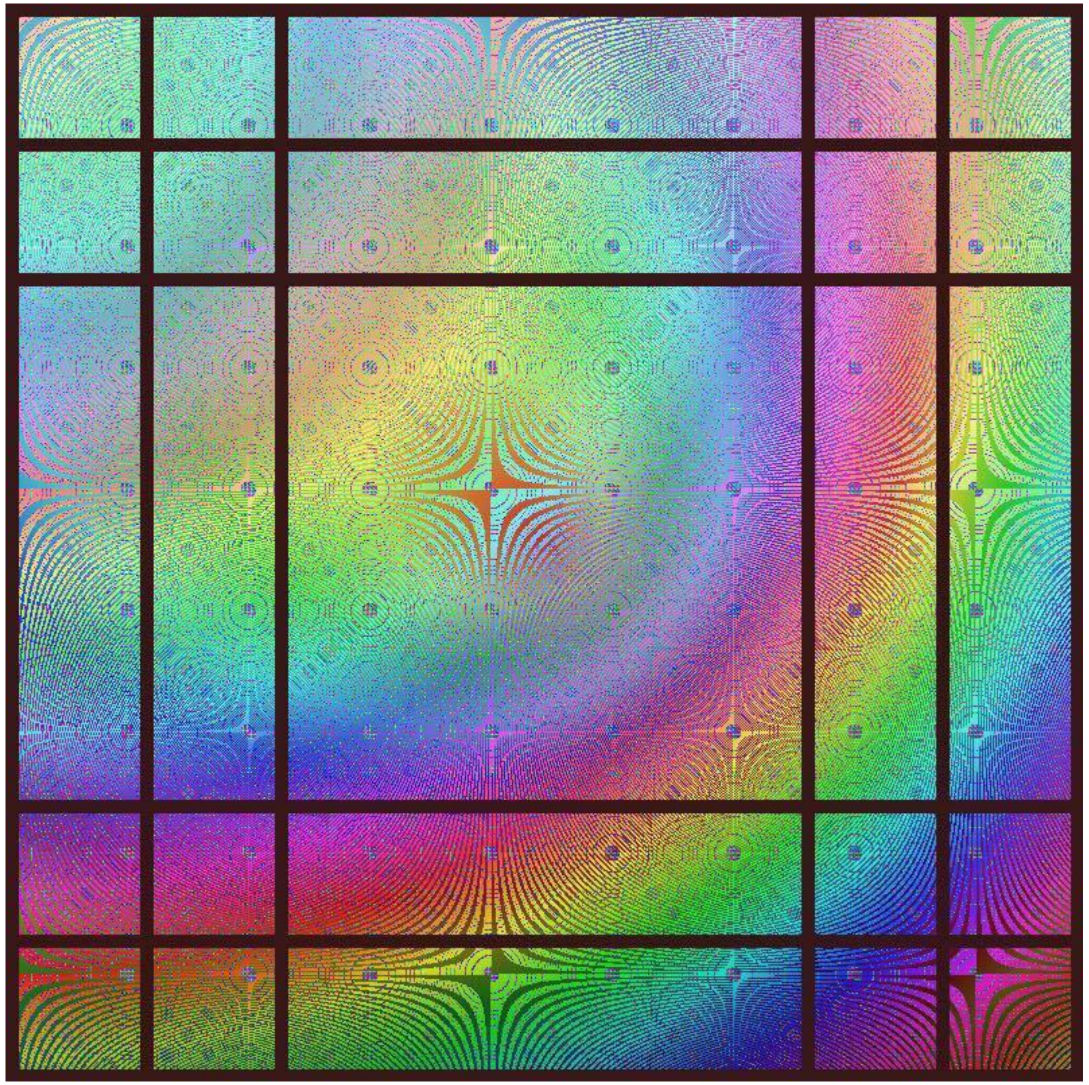




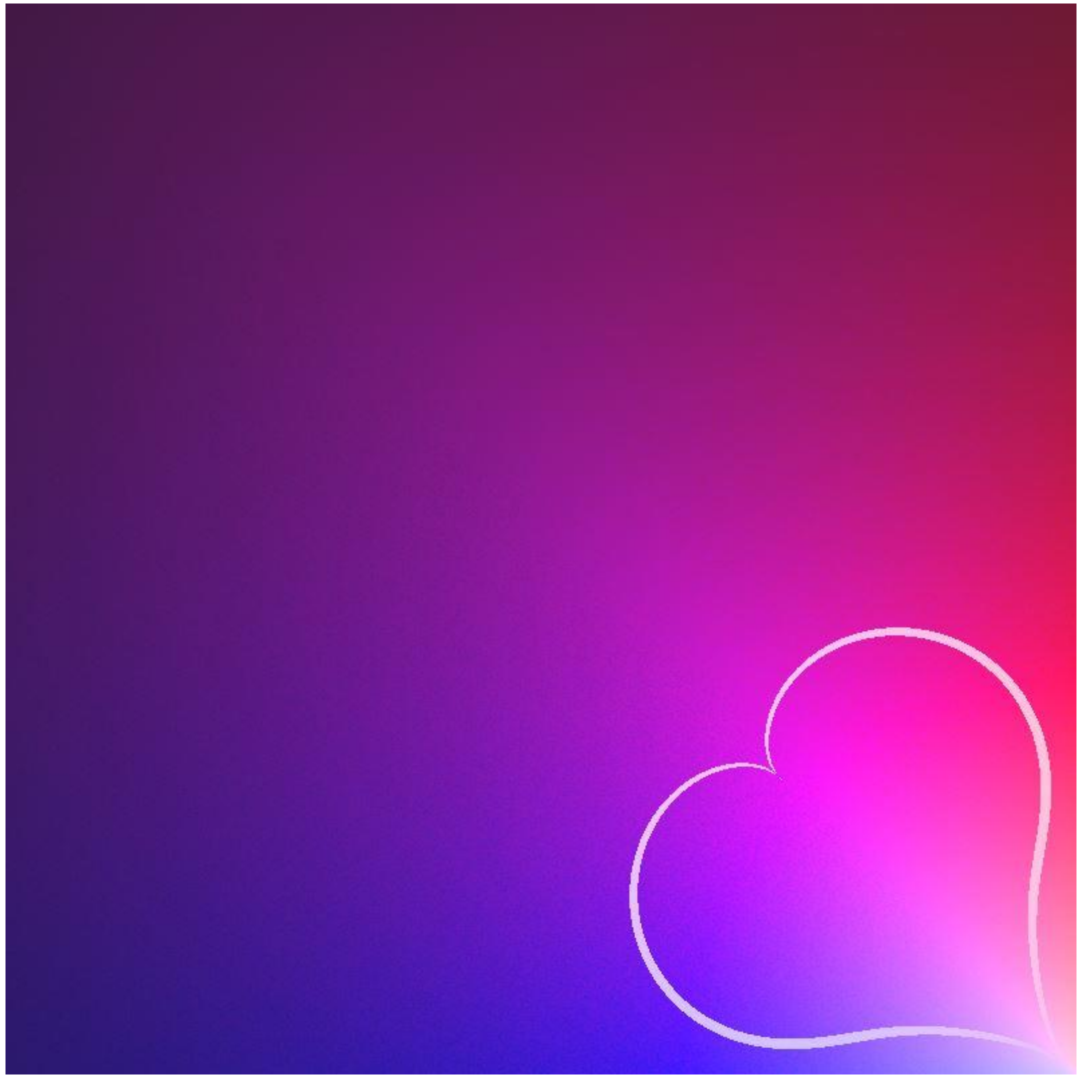
MP1



MP1



MP1



MP1

Due: Monday, Jan. 29th (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 <u>all</u> the artwork after the submission deadline. Course staff and I will give **+1** to all that stand out!

joinSpheres-byValue.cpp

```cpp
/*
 * Creates a new sphere that contains the exact volume
 * of the volume of the two input spheres.
 */
Sphere joinSpheres(Sphere s1, 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;
}
```

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

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

  return 0;
}
```

joinSpheres-byPointer.cpp

```cpp
/*
 * Creates a new sphere that contains the exact volume
 * of the volume of the two input spheres.
 */
Sphere joinSpheres(Sphere * s1, 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;
}
```

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

  Sphere s3 = joinSpheres(s1, s2);

  return 0;
}
```

joinSpheres-byReference.cpp

```cpp
/*
 * Creates a new sphere that contains the exact volume
 * of the volume of the two input spheres.
 */
Sphere joinSpheres(Sphere & s1, 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;
}
```

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

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

  return 0;
}
```

Parameter Passing Properties

	By Value `void foo(Sphere a) { … }`	**By Pointer** `void foo(Sphere *a) { … }`	**By Reference** `void foo(Sphere &a) { … }`
Exactly what is copied when the function is invoked?			
Does modification of the passed in object modify the caller’s object?			
Is there always a valid object passed in to the function?			
Speed			
Programming Safety			

Using `const` in function parameters

joinSpheres-byValue-const.cpp

```cpp
/*
 * 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;
}
```

```cpp
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;
}
```

joinSpheres-byPointer-const.cpp

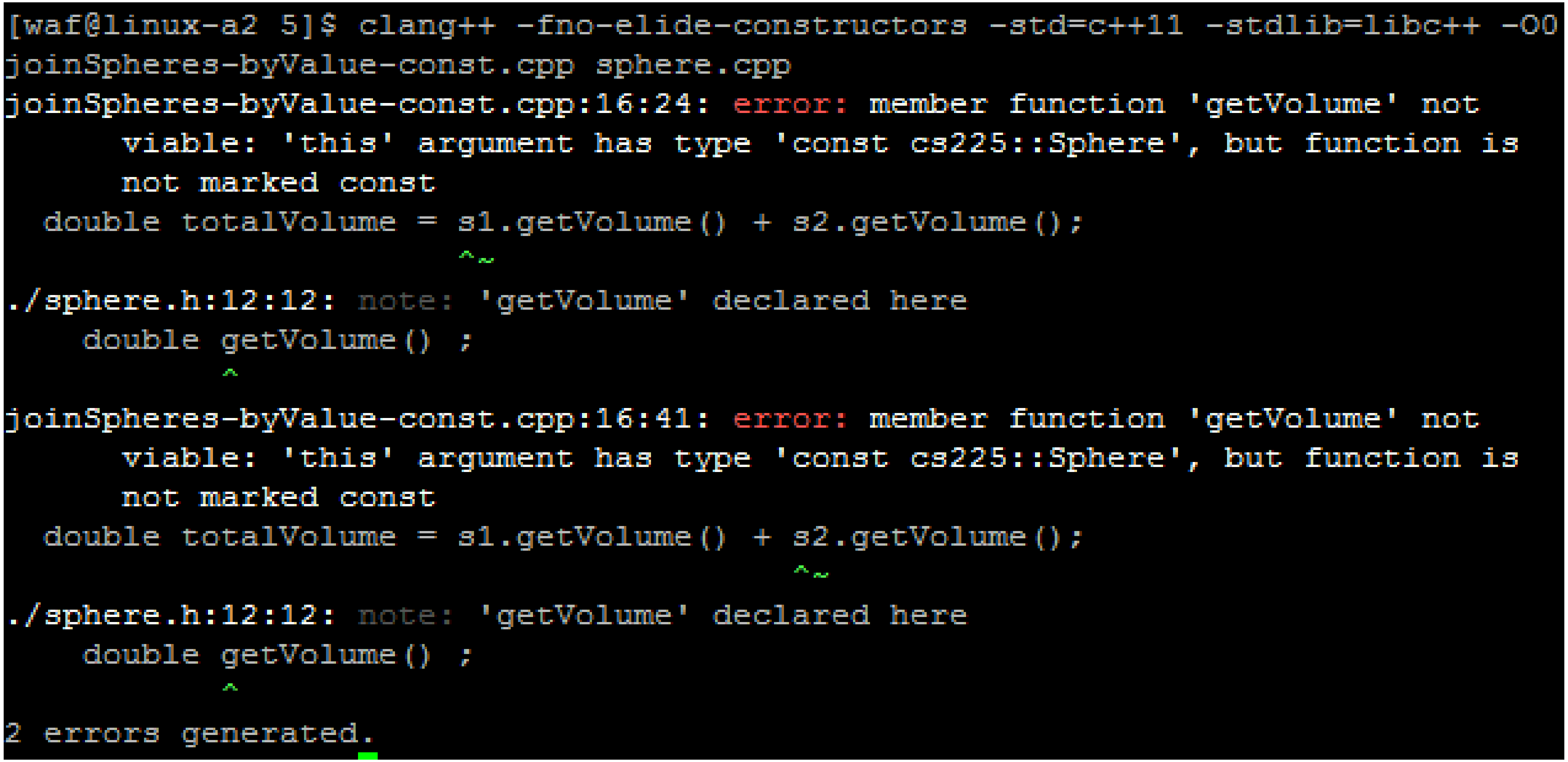
```cpp
/*
 * Creates a new sphere that contains the exact volume
 * of the volume of the two input spheres.
 */
Sphere joinSpheres(Sphere const *s1, Sphere const *s2) {
  double totalVolume = s1->getVolume() + s2->getVolume();

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

  Sphere result(newRadius);

  return result;
}
```

```cpp
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;
}
```

joinSpheres-byReference-const.cpp

```cpp
/*
 * 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;
}
```

```cpp
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;
}
```



```
[waf@linux-a2 5]$ clang++ -fno-elide-constructors -std=c++11 -stdlib=libc++ -O0
joinSpheres-byValue-const.cpp sphere.cpp
joinSpheres-byValue-const.cpp:16:24: error: member function 'getVolume' not
      viable: 'this' argument has type 'const cs225::Sphere', but function is
      not marked const
  double totalVolume = s1.getVolume() + s2.getVolume();
                       ^~
./sphere.h:12:12: note: 'getVolume' declared here
    double getVolume() ;
           ^
joinSpheres-byValue-const.cpp:16:41: error: member function 'getVolume' not
      viable: 'this' argument has type 'const cs225::Sphere', but function is
      not marked const
  double totalVolume = s1.getVolume() + s2.getVolume();
                                        ^~
./sphere.h:12:12: note: 'getVolume' declared here
    double getVolume() ;
           ^
2 errors generated.
```

`const` as part of a member functions’ declaration

sphere.h

```cpp
#ifndef SPHERE_H
#define SPHERE_H

namespace cs225 {
  class Sphere {
    public:
      Sphere();
      Sphere(double r);

      double getRadius();
      double getVolume();

      void setRadius(double r);

    private:
      double r_;

  };
}

#endif
```

sphere.cpp

```cpp
#include "sphere.h"

namespace cs225 {
  Sphere::Sphere() : Sphere(1) { }

  Sphere::Sphere(double r) {
     r_ = r;
  }

  double Sphere::getRadius() {
    return r_;
  }

  double Sphere::getVolume() {
    return (4 * r_ * r_ * r_ *
                3.14159265) / 3.0;
  }

  void setRadius(double r) {
     r_ = r;
  }
}
```

joinSpheres-byValue.cpp

```cpp
/*
 * Creates a new sphere that contains the exact volume
 * of the volume of the two input spheres.
 */
Sphere joinSpheres(Sphere s1, 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;
}
```

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

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

  return 0;
}
```

Copy Constructor

[Purpose]:

All copy constructors will

Copy Constructor

Automatic Copy Constructor

Custom Copy Constructor

sphere.h

```
#ifndef SPHERE_H
#define SPHERE_H

namespace cs225 {
  class Sphere {
    public:
      Sphere(const Sphere & other);
      Sphere();
      Sphere(double r);


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

      void setRadius(double r);

    private:
      double r_;
  };
}

#endif
```

sphere.cpp

```
#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;
  }







  // ...
```

joinSpheres-byValue-const.cpp

```
/*
 * 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;
}
```

Calls to constructors

	By Value `void foo(Sphere a) { … }`	**By Pointer** `void foo(Sphere *a) { … }`	**By Reference** `void foo(Sphere &a) { … }`
`Sphere::Sphere()`			
`Sphere::Sphere(double)`			
`Sphere::Sphere(const Sphere&)`			

joinSpheres-byPointer-const.cpp

```cpp
/*
 * Creates a new sphere that contains the exact volume
 * of the volume of the two input spheres.
 */
Sphere joinSpheres(Sphere const *s1, Sphere const *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;
}
```

```cpp
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;
}
```

joinSpheres-byReference-const.cpp

```cpp
/*
 * 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;
}
```

```cpp
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;
}
```

CS 225 – Things To Be Doing

Register for Theory Exam 1 (CBTF)

More Info: https://courses.engr.illinois.edu/cs225/sp2018/exams/

Complete lab_debug

Due on Sunday at 11:59pm

Finish MP1 – Due Monday

Due on Monday

MP2 Released on Tuesday – Up to +7 Extra Credit for Early Submission

POTD

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