```cpp
int *x;
int size = 3;
x = new int[size];
for (int i = 0; i < size; i++) {
x[i] = i + 3;
}
delete[] x;
```
Pointers and References

A variable containing an instance of an object:

```
Cube s1;
```

A reference variable of a Cube object:

```
Cube & r1 = s1;
```

A variable containing a pointer to a Cube object:

```
Cube * p1;
```
Reference Variable

Three facts about reference variables:

1.

2.

3.
Reference Variable

A reference variable is an alias to an existing variable.

```cpp
#include <iostream>

int main() {
    int i = 7;
    int & j = i; // j is an alias of i
    j = 4;
    std::cout << i << " " << j << std::endl;
    i = 2;
    std::cout << i << " " << j << std::endl;
    return 0;
}
```
Problems of the Day (PotD)

• Log on to PrairieLearn join CS 225 FA21 Course
• The first PotD is out now and due tomorrow at noon
• 100% on it gets 1 point of extra credit
• Max 40 points of extra credit from PotDs and GDBotW
• New PotDs every business day (not weekends or holidays)
• Approximately 60 problems in the week.
/*
 * 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 Value (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>Exactly what is copied when the function is invoked?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Does modification of the passed in object modify the caller’s object?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is there always a valid object passed in to the function?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Speed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programming Safety</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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;
}
const as part of a member functions' declaration
#pragma once

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

#include "Cube.h"

namespace cs225 {
    Cube::Cube() {
        length_ = 1;
    }
    Cube::Cube(double length) {
        length_ = length;
    }
    double Cube::getVolume() {
        return length_ * length_ * length_;
    }
    double Cube::getSurfaceArea() {
        return 6 * length_ * length_;
    }
}
Returning Pointers and References

A variable containing an instance of an object:

```
Cube joinCubes(const Cube &s1, const Cube &s2)
```

A reference variable of a Cube object:

```
Cube &joinCubes(const Cube &s1, const Cube &s2)
```

A variable containing a pointer to a Cube object:

```
Cube *joinCubes(const Cube &s1, const Cube &s2)
```
## Calls to constructors

<table>
<thead>
<tr>
<th></th>
<th>By Value</th>
<th>By Value (Pointer)</th>
<th>By Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>void foo(Cube a)</td>
<td>void foo(Cube *)</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>
Copy Constructor

[Purpose]:
All copy constructors will
Copy Constructor

Automatic Copy Constructor

Custom Copy Constructor