CS 225
Data Structures

September 3 - Lifecycle
G Carl Evans
## 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
```cpp
#pragma once

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

    private:
      double length_;  
  }
}
```

```cpp
#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:

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

A reference variable of a Cube object:

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

A variable containing a pointer to a Cube object:

```cpp
Cube *joinCubes(const Cube &s1, const Cube &s2)
```
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></th>
<th>By Value void foo(Cube a) { ... }</th>
<th>By Pointer void foo(Cube *a) { ... }</th>
<th>By Reference void foo(Cube &amp;a) { ... }</th>
</tr>
</thead>
<tbody>
<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(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;
}
#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_; 
    Cube *ptr_; 
    const Cube &ref; 
};
Tower::Tower(const Tower & other) {
    cube_ = other.cube_;  
    ptr_ = other.ptr_;    
    ref_ = other.ref_;    
    }

```cpp
Tower::Tower(const Tower & other) {
    cube_ = other.cube_;  
    ptr_ = other.ptr_;   
    ref_ = other.ref_;   
}
```

```
waf@seibl-2215-02:/mnt/c/Users/waf/Desktop/cs225/lecture/06-lifecycle$ make
clang++ -std=c++1y -stdlib=libc++ -00 -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_; 
    ^ ~~~~~~~~~~~
```

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_) {}
Tower::Tower(const Tower & other) {
    // Deep copy cube_

    // Deep copy ptr_

    // Deep copy ref_
}
Destructor
```cpp
#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_;  
    }
}
```

```cpp
namespace cs225 {
    Cube::Cube() {
        length_ = 1;
        cout << "Default ctor" << endl;
    }

    Cube::Cube(double length) {
        length_ = length;
        cout << "1-arg ctor" << endl;
    }
    ...
    // ...
}
```
<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>
#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() {
    cout << "dtor called";
    << endl;
}

... // ...