CS 225

Data Structures

January 26 – C++ Parameters

G Carl Evans
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>Exactly what is copied when the function is invoked?</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>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_;  
};

}
Returning Pointers and References

A variable containing an instance of an object:

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

A reference variable of a Cube object:

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

A variable containing a pointer to a Cube object:

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

<table>
<thead>
<tr>
<th>By Value</th>
<th>By Value (Pointer)</th>
<th>By Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void foo(Cube a) { ... }</code></td>
<td><code>void foo(Cube *a) { ... }</code></td>
<td><code>void foo(Cube &amp;a) { ... }</code></td>
</tr>
</tbody>
</table>

- **Cube::Cube()**
- **Cube::Cube(double)**
- **Cube::Cube(const Cube &)**
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</th>
<th>By Pointer</th>
<th>By Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>void foo(Cube a) { ... }</code></td>
<td><code>void foo(Cube *a) { ... }</code></td>
<td><code>void foo(Cube &amp;a) { ... }</code></td>
</tr>
<tr>
<td><code>Cube::Cube()</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Cube::Cube(double)</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Cube::Cube(const Cube&amp;)</code></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;
};
```cpp
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) {
    cube_ = other.cube_;  
    ptr_ = other.ptr_;    
    ref_ = other.ref_;    
}

Tower::Tower(const Tower & other) : cube_(other.cube_),  
                                  ptr_(other.ptr_), ref_(other.ref_) {}
```cpp
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><strong>Bitwise</strong></td>
</tr>
<tr>
<td><strong>Assignment</strong></td>
</tr>
<tr>
<td><strong>Comparison</strong></td>
</tr>
<tr>
<td><strong>Logical</strong></td>
</tr>
<tr>
<td><strong>Other</strong></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;
    }
    // ...
}