



# CS 225

## Data Structures

*January 31 - Parameters*

*G Carl Evans*

## heap-puzzle3.cpp

```
5 int *x;
6 int size = 3;
7
8 x = new int[size];
9
10 for (int i = 0; i < size; i++) {
11     x[i] = i + 3;
12 }
13
14 delete[] x;
```

## joinCubes-byValue.cpp

```
11  /*
12  * Creates a new Cube that contains the exact volume
13  * of the volume of the two input Cubes.
14  */
15  Cube joinCubes(Cube c1, Cube c2) {
16      double totalVolume = c1.getVolume() + c2.getVolume();
17
18      double newLength = std::pow( totalVolume, 1.0/3.0 );
19
20      Cube result(newLength);
21      return result;
22  }
```

```
23
24
25
26
28  int main() {
29      Cube *c1 = new Cube(4);
30      Cube *c2 = new Cube(5);
31
32      Cube c3 = joinCubes(*c1, *c2);
33
34      return 0;
35  }
```

## joinCubes-byPointer.cpp

```
11  /*
12   * Creates a new Cube that contains the exact volume
13   * of the volume of the two input Cubes.
14   */
15  Cube joinCubes(Cube * c1, Cube * c2) {
16      double totalVolume = c1->getVolume() + c2->getVolume();
17
18      double newLength = std::pow( totalVolume, 1.0/3.0 );
19
20      Cube result(newLength);
21      return result;
22  }
23
24
25
26
```

```
28  int main() {
29      Cube *c1 = new Cube(4);
30      Cube *c2 = new Cube(5);
31
32      Cube c3 = joinCubes(c1, c2);
33
34      return 0;
35  }
```

## joinCubes-byRef.cpp

```
11  /*
12  * Creates a new Cube that contains the exact volume
13  * of the volume of the two input Cubes.
14  */
15  Cube joinCubes(Cube & c1, Cube & c2) {
16      double totalVolume = c1.getVolume() + c2.getVolume();
17
18      double newLength = std::pow( totalVolume, 1.0/3.0 );
19
20      Cube result(newLength);
21      return result;
22  }
```

```
23
24
25
26
28  int main() {
29      Cube *c1 = new Cube(4);
30      Cube *c2 = new Cube(5);
31
32      Cube c3 = joinCubes(*c1, *c2);
33
34      return 0;
35  }
```

# Parameter Passing Properties

	<b>By Value</b> <code>void foo(Cube a) { ... }</code>	<b>By Pointer</b> <code>void foo(Cube *a) { ... }</code>	<b>By Reference</b> <code>void foo(Cube &amp;a) { ... }</code>
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			



# MP1

**Due:** Wednesday, Feb. 5<sup>th</sup> (11:59pm)

**Share your art work:**

- On our piazza, in the “MP1 - Artwork Sharing” thread



Using `const` in function parameters



## joinCubes-byValue-const.cpp

```
11  /*
12  * Creates a new Cube that contains the exact volume
13  * of the volume of the two input Cubes.
14  */
15  Cube joinCubes(const Cube c1, const Cube c2) {
16      double totalVolume = c1.getVolume() + c2.getVolume();
17
18      double newLength = std::pow( totalVolume, 1.0/3.0 );
19
20      Cube result(newLength);
21      return result;
22  }
```

```
23
24
25
26
```

```
28  int main() {
29      Cube *c1 = new Cube(4);
30      Cube *c2 = new Cube(5);
31
32      Cube c3 = joinCubes(*c1, *c2);
33
34      return 0;
35  }
```

## joinCubes-byPointer-const.cpp

```
11  /*
12  * Creates a new Cube that contains the exact volume
13  * of the volume of the two input Cubes.
14  */
15  Cube joinCubes(const Cube * c1, const Cube * c2) {
16      double totalVolume = c1->getVolume() + c2->getVolume();
17
18      double newLength = std::pow( totalVolume, 1.0/3.0 );
19
20      Cube result(newLength);
21      return result;
22  }
23
24
25
26
```

```
28  int main() {
29      Cube *c1 = new Cube(4);
30      Cube *c2 = new Cube(5);
31
32      Cube c3 = joinCubes(c1, c2);
33
34      return 0;
35  }
```

## joinCubes-byRef-const.cpp

```
11  /*
12  * Creates a new Cube that contains the exact volume
13  * of the volume of the two input Cubes.
14  */
15  Cube joinCubes(const Cube & c1, const Cube & c2) {
16      double totalVolume = c1.getVolume() + c2.getVolume();
17
18      double newLength = std::pow( totalVolume, 1.0/3.0 );
19
20      Cube result(newLength);
21      return result;
22  }
```

```
23
24
25
26
```

```
28  int main() {
29      Cube *c1 = new Cube(4);
30      Cube *c2 = new Cube(5);
31
32      Cube c3 = joinCubes(*c1, *c2);
33
34      return 0;
35  }
```

TERMINAL

1: wsl

```
waf@siebl-2215-02:/mnt/c/Users/waf/Desktop/cs225/_lecture/05-parameters$  
make  
clang++ -std=c++1y -stdlib=libc++ -O0 -Wall -Wextra -pedantic -lpthread -  
lm joinCubes-byValue-const.cpp cs225/Cube.cpp -lm -o joinCubes-byValue-co  
nst  
joinCubes-byValue-const.cpp:16:24: error: member function 'getVolume' not  
viable: 'this' argument has type 'const cs225::Cube', but function is no  
t marked const  
    double totalVolume = c1.getVolume() + c2.getVolume();  
                          ^~  
./cs225/Cube.h:9:14: note: 'getVolume' declared here  
    double getVolume();  
      ^  
joinCubes-byValue-const.cpp:16:41: error: member function 'getVolume' not  
viable: 'this' argument has type 'const cs225::Cube', but function is no  
t marked const  
    double totalVolume = c1.getVolume() + c2.getVolume();  
                          ^~  
./cs225/Cube.h:9:14: note: 'getVolume' declared here  
    double getVolume();  
      ^  
2 errors generated.  
Makefile:19: recipe for target 'joinCubes-byValue-const' failed  
make: *** [joinCubes-byValue-const] Error 1  
waf@siebl-2215-02:/mnt/c/Users/waf/Desktop/cs225/_lecture/05-parameters$  
█
```



**const** as part of a member functions' declaration

## Cube.h

```
1 #pragma once
2
3 namespace cs225 {
4     class Cube {
5     public:
6         Cube();
7         Cube(double length);
8         double getVolume();
9         double getSurfaceArea();
10
11     private:
12         double length_;
13     };
14 }
15
16
17
18
19
20
```

## Cube.cpp

```
1 #include "Cube.h"
2 namespace cs225 {
3     Cube::Cube() {
4         length_ = 1;
5     }
6
7     Cube::Cube(double length) {
8         length_ = length;
9     }
10
11     double Cube::getVolume() {
12         return length_ * length_ *
13             length_;
14     }
15
16     double
17     Cube::getSurfaceArea() {
18         return 6 * length_ *
19             length_;
20     }
21 }
```

## joinCubes-byValue-const.cpp

```
11  /*
12  * Creates a new Cube that contains the exact volume
13  * of the volume of the two input Cubes.
14  */
15  Cube joinCubes(const Cube c1, const Cube c2) {
16      double totalVolume = c1.getVolume() + c2.getVolume();
17
18      double newLength = std::pow( totalVolume, 1.0/3.0 );
19
20      Cube result(newLength);
21      return result;
22  }
```

```
23
24
25
26
```

```
28  int main() {
29      Cube *c1 = new Cube(4);
30      Cube *c2 = new Cube(5);
31
32      Cube c3 = joinCubes(*c1, *c2);
33
34      return 0;
35  }
```



# Copy Constructor

**[Purpose]:**

All copy constructors will





# Copy Constructor

**Automatic Copy Constructor**

**Custom Copy Constructor**

## Cube.h

```
1 #pragma once
2
3 namespace cs225 {
4     class Cube {
5     public:
6         Cube();
7         Cube(double length);
8
9         Cube(const Cube & other);
10
11         double getVolume() const;
12         double getSurfaceArea() const;
13
14     private:
15         double length_;
16     };
17 }
18
19
20
```

## Cube.cpp

```
7 namespace cs225 {
8     Cube::Cube() {
9         length_ = 1;
10        cout << "Default ctor"
11            << endl;
12    }
13
14    Cube::Cube(double length) {
15        length_ = length;
16        cout << "1-arg ctor"
17            << endl;
18    }
19
20
21
22
23
24
25
... // ...
```

## joinCubes-byValue-const.cpp

```
11  /*
12  * Creates a new Cube that contains the exact volume
13  * of the volume of the two input Cubes.
14  */
15  Cube joinCubes(const Cube c1, const Cube c2) {
16      double totalVolume = c1.getVolume() + c2.getVolume();
17
18      double newLength = std::pow( totalVolume, 1.0/3.0 );
19
20      Cube result(newLength);
21      return result;
22  }
```

```
23
24
25
26
```

```
28  int main() {
29      Cube *c1 = new Cube(4);
30      Cube *c2 = new Cube(5);
31
32      Cube c3 = joinCubes(*c1, *c2);
33
34      return 0;
35  }
```

# Calls to constructors

	<b>By Value</b> <code>void foo(Cube a) { ... }</code>	<b>By Pointer</b> <code>void foo(Cube *a) { ... }</code>	<b>By Reference</b> <code>void foo(Cube &amp;a) { ... }</code>
<code>Cube::Cube()</code>			
<code>Cube::Cube(double)</code>			
<code>Cube::Cube(const Cube &amp;)</code>			

## joinCubes-byPointer-const.cpp

```
11  /*
12   * Creates a new Cube that contains the exact volume
13   * of the volume of the two input Cubes.
14   */
15  Cube joinCubes(const Cube * c1, const Cube * c2) {
16      double totalVolume = c1->getVolume() + c2->getVolume();
17
18      double newLength = std::pow( totalVolume, 1.0/3.0 );
19
20      Cube result(newLength);
21      return result;
22  }
23
24
25
26
```

```
28  int main() {
29      Cube *c1 = new Cube(4);
30      Cube *c2 = new Cube(5);
31
32      Cube c3 = joinCubes(c1, c2);
33
34      return 0;
35  }
```

## joinCubes-byRef-const.cpp

```
11  /*
12  * Creates a new Cube that contains the exact volume
13  * of the volume of the two input Cubes.
14  */
15  Cube joinCubes(const Cube & c1, const Cube & c2) {
16      double totalVolume = c1.getVolume() + c2.getVolume();
17
18      double newLength = std::pow( totalVolume, 1.0/3.0 );
19
20      Cube result(newLength);
21      return result;
22  }
```

```
23
24
25
26
```

```
28  int main() {
29      Cube *c1 = new Cube(4);
30      Cube *c2 = new Cube(5);
31
32      Cube c3 = joinCubes(*c1, *c2);
33
34      return 0;
35  }
```