# CS 225

**Data Structures**

*September 3 - Lifecycle*

*G Carl Evans*

# Parameter Passing Properties

| | **By Value**<br>`void foo(Cube a) { … }` | **By Value (Pointer)**<br>`void foo(Cube *a) { … }` | **By Reference**<br>`void foo(Cube &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

## 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 }
```

# `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
     Cube::getSurfaceArea(){
17     return 6 * length_ *
18            length_;
19   }
20 }
21
```

# 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)
```

# Copy Constructor

# 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
10        double getVolume() const;
11        double getSurfaceArea() const;
12
13      private:
14        double length_;
15    };
16  }
17
18
19
20
```

## Cube.cpp

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

## 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 }
```

# Calls to constructors

| | By Value<br>`void foo(Cube a) { … }` | By Pointer<br>`void foo(Cube *a) { … }` | By Reference<br>`void foo(Cube &a) { … }` |
|---|---|---|---|
| `Cube::Cube()` | | | |
| `Cube::Cube(double)` | | | |
| `Cube::Cube(const Cube&)` | | | |

## 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 }
```

## Tower.h

```
 1 #pragma once
 2
 3 #include "cs225/Cube.h"
 4 using cs225::Cube;
 5
 6 class Tower {
 7   public:
 8     Tower(Cube c, Cube *ptr, const Cube &ref);
 9     Tower(const Tower & other);
10
11   private:
12     Cube cube_;
13     Cube *ptr_;
14     const Cube &ref;
15 };
16
17
```

Tower.cpp

```
10 Tower::Tower(const Tower & other) {
11   cube_ = other.cube_;
12   ptr_ = other.ptr_;
13   ref_ = other.ref_;
14 }
```

## Tower.cpp

```
10 Tower::Tower(const Tower & other) {
11   cube_ = other.cube_;
12   ptr_ = other.ptr_;
13   ref_ = other.ref_;
14 }
```

```
waf@siebl-2215-02:/mnt/c/Users/waf/Desktop/cs225/_lecture/06-lifecycle$ make
clang++ -std=c++1y -stdlib=libc++ -O0 -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.cpp

```
10 Tower::Tower(const Tower & other) {
11   cube_ = other.cube_;
12   ptr_ = other.ptr_;
13   ref_ = other.ref_;
14 }
```

Tower.cpp

```
10 Tower::Tower(const Tower & other) : cube_(other.cube_),
11    ptr_(other.ptr_), ref_(other.ref_) { }
12
13
14
```

***Constructor Initializer List***

## Tower.cpp

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




  // Deep copy ptr_:




  // Deep copy ref_:




}
```

# Destructor

## Cube.h

```
 1  #pragma once
 2
 3  namespace cs225 {
 4    class Cube {
 5      public:
 6        Cube();
 7        Cube(double length);
 8        Cube(const Cube & other);
 9        ~Cube();
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"
             << endl;
11    }
12
13    Cube::Cube(double length) {
14      length_ = length;
15      cout << "1-arg ctor"
             << endl;
16    }
17
18
19
20
21
22
23
24
25
…     // ...
```

| Operators that can be overloaded in C++ | |
|---|---|
| Arithmetic | + - * / % ++ -- |
| Bitwise | & \| ^ ~ << >> |
| Assignment | = |
| Comparison | == != > < >= <= |
| Logical | ! && \|\| |
| Other | [] () -> |

## Cube.h

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

## Cube.cpp

```
 7  namespace cs225 {
 8    Cube::~Cube() {
 9      cout << "dtor called";
           << endl;
10    }
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
…     // ...
```