



CS 225

Data Structures

September 4 - Lifecycle

G Carl Evans



Exam 1

- No stream next Friday Exam 1 is during lecture
- Covers pre-req theory material and basic C++
 - More info here <https://courses.engr.illinois.edu/cs225/fa2020/exams/exam1/>
- Practice Exam available later today
- Exam using CBTF Online

CBTF Online

Online proctoring with Zoom and CBTF proctors

How does it work?

1. Make a reservation in the [CBTF Scheduler](#).
2. Visit the Scheduler near your reservation time to check in and receive further instructions, including the link to the Zoom meeting for your proctoring.
3. Join the Zoom meeting from your phone and position it to view you taking your exam. The proctor is present to help get you set up, answer questions, etc.
4. Once checked in, you will be given instructions via the Scheduler on how to take the exam.

CBTF Online

Visit cbtf.engr.illinois.edu

- Full instructions
- FAQ
- Walkthrough of check-in process
- Conflict exam request form
- CBTF Scheduler

Still have questions or concerns? Chat, ask questions, and get to know the proctors during our regularly scheduled Fall 2020 [CBTF Proctor Office Hours](#):

- Mon, 5:30 pm - 6:30 pm CST
- Tue, 12:00 pm - 1:00 pm CST
- Wed, 5:30 pm - 6:30 pm CST
- Thu, 7:30 pm - 8:30 pm CST
- Fri, 12:00 pm - 1:00 pm CST
- Sat, 5:30 pm - 6:30 pm CST
- Sun, 7:30 pm - 8:30 pm CST



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

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"
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
... // ...
```

Operators that can be overloaded in C++

Arithmetic	<code>+</code>	<code>-</code>	<code>*</code>	<code>/</code>	<code>%</code>	<code>++</code>	<code>--</code>
Bitwise	<code>&</code>	<code> </code>	<code>^</code>	<code>~</code>	<code><<</code>	<code>>></code>	
Assignment	<code>=</code>						
Comparison	<code>==</code>	<code>!=</code>	<code>></code>	<code><</code>	<code>>=</code>	<code><=</code>	
Logical	<code>!</code>	<code>&&</code>	<code> </code>				
Other	<code>[]</code>	<code>()</code>	<code>-></code>				

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";
10        << endl;
11    }
12
13
14
15
16
17
18
19
20
21
22
23
24
25
... // ...
```