CS 225

Data Structures

February 3- 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;
```

Pointers and References

A variable containing an instance of an object:

```
1 Cube s1;
```

A reference variable of a Cube object:

```
1 Cube & r1 = s1;
```

A variable containing a pointer to a Cube object:

```
1 Cube * p1;
```

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

	By Value `void foo(Cube a) { … }`	**By Value (Pointer)** `void foo(Cube *a) { … }`	**By Reference** `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
```

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"
            << 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-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

	By Value `void foo(Cube a) { … }`	By Value (Pointer) `void foo(Cube *a) { … }`	By Reference `void foo(Cube &a) { … }`
`Cube::Cube()`			
`Cube::Cube(double)`			
`Cube::Cube(const Cube &)`			

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 }
```