



CS 225

Data Structures

January 24 – C++ Review

G Carl Evans

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;
```



Pointers

Three key ideas:

1.

2.

3.



Indirection Operators

Given any variable **v**:

&v

***v**

v->

main.cpp

```
1 #include <iostream>
2 #include "Cube.h"
3
4 int main() {
5     cs225::Cube c;
6     std::cout << "Address storing `c`:" << &c << std::endl;
7
8     cs225::Cube *ptr = &c;
9     std::cout << "Addr. storing ptr: " << &ptr << std::endl;
10    std::cout << "Contents of ptr: " << ptr << std::endl;
11
12    return 0;
13 }
```



Heap Memory - new

As programmers, we can use heap memory in cases where the lifecycle of the variable exceeds the lifecycle of the function.

The only way to create heap memory is with the use of the **new** keyword. Using **new** will:

- 1.

- 2.

- 3.



Heap Memory - delete

2. The only way to free heap memory is with the use of the **delete** keyword. Using **delete** will:

-
-

3. Memory is never automatically reclaimed, even if it goes out of scope. Any memory lost, but not freed, is considered to be “leaked memory”.

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;
```




Heap Memory vs. Stack Memory Lifecycle



Reference Variable

A reference variable is an alias to an existing variable.

Key Idea: Modifying the reference variable modifies the variable being aliased.



Reference Variable

Three facts about reference variables:

1.

2.

3.

Reference Variable

A reference variable is an alias to an existing variable.

```
1 #include <iostream>
2
3 int main() {
4     int i = 7;
5     int & j = i;    // j is an alias of i
6
7     j = 4;
8     std::cout << i << " " << j << std::endl;
9
10    i = 2;
11    std::cout << i << " " << j << std::endl;
12    return 0;
13 }
```

heap-puzzle1.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int *x = new int;
6     int &y = *x;
7
8     y = 4;
9
10    cout << &x << endl;
11    cout << x << endl;
12    cout << *x << endl;
13
14    cout << &y << endl;
15    cout << y << endl;
16    cout << *y << endl;
17 }
```

heap-puzzle2.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int *p, *q;
6     p = new int;
7     q = p;
8     *q = 8;
9     cout << *p << endl;
10
11     q = new int;
12     *q = 9;
13     cout << *p << endl;
14     cout << *q << endl;
15
16     return 0;
17 }
```

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