Puzzle from last Friday:

```
puzzle.cpp
4 Cube *CreateCube() {
5     Cube c(20);
6     return &c;
7 }
8
9 int main() {
10    Cube *c = CreateCube();
11    SomeOtherFunction();
12    double v = c->getVolume();
13    double a = c->getSurfaceArea();
14    return 0;
15 }
```

Takeaway:

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

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

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”.

```
heap1.cpp
8 int main() {
9    int *p = new int;
10   cs225::Cube *c = new cs225::Cube(10);
11   return 0;
12 }
```

```
heap2.cpp
4 int main() {
5    Cube *c1 = new Cube();
6    Cube *c2 = c1;
7    c2->setLength( 10 );
8    delete c2;
9    delete c1;
10   return 0;
11 }
```
Copying Memory – Deep Copy vs. Shallow Copy

```
copy.cpp
6  int i = 2, j = 4, k = 8;
7  int *p = &i, *q = &j, *r = &k;
8  k = i;
9  cout << i << j << k << *p << *q << *r << endl;
12  p = q;
13  cout << i << j << k << *p << *q << *r << endl;
15  *q = *r;
16  cout << i << j << k << *p << *q << *r << endl;
```

Consider how each assignment operator changes the data:

<table>
<thead>
<tr>
<th>Type of LHS</th>
<th>Type of RHS</th>
<th>Data Changed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line 8-9</td>
<td></td>
<td>i =</td>
</tr>
<tr>
<td></td>
<td></td>
<td>j =</td>
</tr>
<tr>
<td></td>
<td></td>
<td>k =</td>
</tr>
<tr>
<td></td>
<td></td>
<td>p =</td>
</tr>
<tr>
<td></td>
<td></td>
<td>q =</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r =</td>
</tr>
<tr>
<td>Line 11-12</td>
<td></td>
<td>i =</td>
</tr>
<tr>
<td></td>
<td></td>
<td>j =</td>
</tr>
<tr>
<td></td>
<td></td>
<td>k =</td>
</tr>
<tr>
<td></td>
<td></td>
<td>p =</td>
</tr>
<tr>
<td></td>
<td></td>
<td>q =</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r =</td>
</tr>
<tr>
<td>Line 14-15</td>
<td></td>
<td>i =</td>
</tr>
<tr>
<td></td>
<td></td>
<td>j =</td>
</tr>
<tr>
<td></td>
<td></td>
<td>k =</td>
</tr>
<tr>
<td></td>
<td></td>
<td>p =</td>
</tr>
<tr>
<td></td>
<td></td>
<td>q =</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r =</td>
</tr>
</tbody>
</table>

Reference Variable

A reference variable is an alias to an existing variable. Modifying the reference variable modifies the variable being aliased. Internally, a reference variable maps to the same memory as the variable being aliased. Three key ideas:

1.
2.
3.

```
reference.cpp
3  int main() {
4    int i = 7;
5    int & j = i; // j is an alias of i
6    j = 4;       // j and i are both 4.
7    std::cout << i << " " << j << std::endl;
9    i = 2;       // j and i are both 2.
11   std::cout << i << " " << j << std::endl;
12   return 0;
13 }
```

```
heap-puzzle1.cpp
6  int *x = new int;
7  int &y = *x;
8  y = 4;
9  cout << &x << endl;
10  cout << x << endl;
11  cout << *x << endl;
12  cout << &y << endl;
13  cout << y << endl;
14  cout << *y << endl;
```

```
heap-puzzle2.cpp
6  int *p, *q;
7  p = new int;
8  q = p;
9  *q = 8;
10  cout << *p << endl;
11  cout << *q << endl;
12  q = new int;
13  *q = 9;
14  cout << *p << endl;
15  cout << *q << endl;
```

CS 225 – Things To Be Doing:

1. Exam 0 starts on Thursday, know your time slot!
2. Finish up MP1 – Due Monday, Sept. 10 at 11:59pm
3. Complete lab_debug this week in lab sections (due Sunday)
4. POTDs are released daily, worth +1 extra credit point! ☺