Recursion, pt. 3:

The Towers of Hanoi
Towers of Hanoi

- One classic recursive problem is that of the Towers of Hanoi.
Towers of Hanoi

- The goal in this problem: to move the entire tower from the left-most peg to the right-most peg.
Towers of Hanoi

• Rules:
  1. A larger ring may never be on top of a smaller ring.
  2. Only one ring may be moved at a time.
Towers of Hanoi

• Rules:

3. Rings may only be removed from the top of one tower and placed on the top of a second tower.
Towers of Hanoi

• Given these rules, how can we solve the problem?
Towers of Hanoi

- To move the tower fully to the right...
Recursive Step

• we’ll have to go through this stage.
Towers of Hanoi

- The most simple way to reach that stage is from this one...
Towers of Hanoi

- ... which can be reached from this stage, and so forth.
Recursive Step

- What common property do these moves have?
  - Our goal is first to move the bottom-most ring of the stack to the destination tower.
  - To do this, we must move the rest of the stack to the spare tower.
Recursive Step

• Note what we’re saying here.
  – “To do this, we must move the rest of the stack to the spare tower.”

• In other words, to move the entire stack from the left to the right tower... will require the ability to move almost the entire stack from the left to the spare tower.
Recursive Step

• To move the stack of rings... requires being able to move (a smaller portion of) the stack.

• Note that this involves a reduction – our stack size will reduce by one upon each such reduction.
Recursive Step

• Combining these two facts together, we have a *perfect* candidate for recursion.

  – We can solve the problem in terms of a reduced form of itself.

  – We’ll move the stack from one tower to another by... moving (most of) the stack from one tower to another. *Recursion.*
Recursive Step

• So, in order...

1. Move the top n-1 rings from the source to the spare tower.

2. Move the bottom (the leftover 1 ring) from the source to the destination tower.

3. Relocate those n-1 rings we had placed on the spare tower over to the destination tower.
Recursive Step

- Note how the process works for this case...
Recursive Step

• ...as well as for this case.
Recursive Step

• ...as well as for this case.
  - The difference here is that the “destination” tower is different here than it was for the other situation.
Recursive Step

- So, while we know the general idea for reducing the problem, there’s the problem of towers serving different purposes at different stages.
Recursive Step

• In order to move the full tower from the left peg to the right...
Recursive Step

• ...we move the top $n-1$ to the *spare* tower.
  – Thus, the spare and destination towers get interchanged, conceptually.
Towers of Hanoi

• The method needs to access all of the towers at every frame.
  – What changes is the name by which we will reference each tower.
  – Hmm. Reference.

• Remember, the changes to each tower should propagate across every frame of the method.
Pseudocode

```c
void hanoi(int n, Tower* a, Tower* b, Tower* c)
{
    hanoi(n-1, a, c, b);
    Ring p = a->takeRing();
    c->placeRing(p);
    hanoi(n-1, b, a, c);
}
```

• Note: “Ring” and “Tower” haven’t been defined or analyzed yet.
References

• Will this work?
  – This is part of the beauty of passing objects by reference. Changes to each tower will persist, and the name by which we call each Tower will change appropriately for the method.
This is a major benefit of using pointers!

- The whole “by reference” thing can be troublesome when starting out, since it can cause collateral damage.

- However, when references are fully understood, that “collateral” damage may be easily focused, channeled, and controlled for great benefit.
Towers of Hanoi

• What’s left to examine?
  – Ah yes. “Ring” and “Tower” still haven’t yet been defined.
  – Let’s look at the towers first.
Towers of Hanoi

• What would be a proper representation for each tower?
  – We are placing rings on the top of each tower and removing them from the top.
Towers of Hanoi

- The towers are effectively stacks!
  - Yeah, there’s a reason we’ve been talking about “stacks of rings.”
Towers of Hanoi

• The towers are effectively stacks!
  – We can model this by using a vector; elements should only be removed from and added at the vector’s end.
void hanoi(int n, Tower* a, Tower* b, Tower* c)
{
    hanoi(n-1, a, c, b);
    c->placeRing(a->takeRing());
    hanoi(n-1, b, a, c);
}

• Wait... something’s missing...
  – What’s our base case?
The Core Algorithm

```c
void hanoi(int n, Tower* a, Tower* b, Tower* c)
{
    if(n > 1)
        hanoi(n-1, a, c, b);
    c->placeRing(a->takeRing());
    if(n > 1)
        hanoi(n-1, b, a, c);
}
```