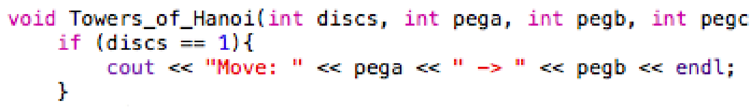
Alex Maass & Sam Odle
CS2024 - C++ Programming
Assignment #3
Deitel & Deitel, Exercise 6.38

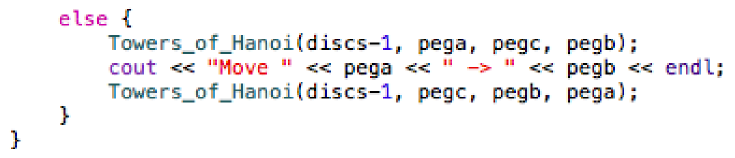
For this assignment we had to write a recursive solution to the classic "Towers of Hanoi" problem. More specifically, we had to write a program that tells how to move different sized discs from a first peg to a third peg, using the second peg as a temporary holding space. The discs can never be arranged such that a larger disc is on top of a smaller disc.

To solve this problem we closely followed the guidelines laid out in the problem. Our function, `Towers_of_Hanoi`, has four parameters: the number of discs to be moved, the starting peg, the ending peg, and the intermediary peg. We start, as shown below and as taught in lecture, by considering the simplest case - where discs = 1. In this case we clearly only need to move the disk from where it is to its destination.



```
void Towers_of_Hanoi(int discs, int pega, int pegb, int pegc){
    if (discs == 1){
        cout << "Move: " << pega << " -> " << pegb << endl;
    }
```

With more rings the problem seems significantly more complicated until you realize that every case from two rings to n rings can be solved with the same series of steps. Thinking generally, if peg three is our ultimate destination, we first call `Towers_of_Hanoi` moving n-1 discs to peg 2 with peg 3 as the holding area.



```
    else {
        Towers_of_Hanoi(discs-1, pega, pegc, pegb);
        cout << "Move " << pega << " -> " << pegb << endl;
        Towers_of_Hanoi(discs-1, pegc, pegb, pega);
    }
}
```

We then call `Towers_of_Hanoi` to move n-1 discs from destination of the first call (the new holding area) to the final destination. These steps can be seen in the code above. Even though it is obvious, it is important to note that when we say 'move' we aren't moving anything. In other words, we aren't explicitly tracking how many discs are on each peg at a given step, we are simply printing out the instructions for solving the problem. You could determine this other information implicitly from the output, but for this assignment it doesn't matter.

Our main function for this assignment prompts the user to input, as you might expect, the number of disks they want to move, the starting peg, the holding peg, and the final peg. There is no check to only let the user enter a certain peg number once because we are assuming any user of our function will have a fundamental understanding of the problem. Implementing such a check wouldn't be hard, but it would clutter our otherwise simple and straightforward execution of this assignment. (There won't be any error if you enter 1 for each of the pegs. The function assumes there are three pegs called peg 1, but your output will be a meaningless stream of 1 -> 1s.)

In closing, this assignment again demonstrated how elegant and simple recursion can be. It also demonstrated how frustrating it can be to try to think through all the steps of a recursive solution to a problem by simply looking at it.