Today we are finishing up our exploration of recursion. Last class we saw an introduction to the Towers of Hanoi problem, we considered how we could break it down into smaller pieces, we discussed intuition on solving the puzzle for 3 disks, and discussed some sample code.

Today we will consider how we can trace recursive code to help us understand how it works.

Questions from last class:

1) If the legend is true and the monks can move a disc from one needle to another in the space of a second, how much longer do we have on Earth?

2) And to help us answer the above question: What is the relationship between the number of disks \( n \) and the number of moves required?

Let's look at code to solve this problem. Compare this to the intuitive steps written down above.

```c
1) // Display extra spaces, to help illustrate the levels of recursion
2) void printSpace( int level)
3) {
4) for( int i=0; i<level; i++) {
5) cout << " ";
6) }
7) }
8)
9)
10) // Move diskToMove from the source tower A to the destination tower B
11) void moveDisk( int diskToMove, char source, char destination, char spare, int level)
12) {
13) if only one disk
14) move disk from source to dest.
15) source dest space
16) move 2 from A to C B
17) move 3 from A to B C
18) move 2 from C to B A
```

Intuitive steps:
For the three disk example above, the disks are numbered as:

```
1) Move diskToMove from the source tower to the destination tower
2) Recursive base case
3) Recursive case
```

The above code has been compacted to facilitate using it in a trace below, where we've removed the portions that prepend extra spaces to reflect the recursion level.

```
void moveDisk( int diskToMove, char source, char destination, char spare, int level)
{
    if( diskToMove == 1 ) {
        printSpace( level);
        cout << "Move disk " << diskToMove << " from " << source << " to " << destination << endl;
    } else {
        moveDisk( diskToMove-1, source, spare, destination, level+1);
        printSpace( level);
        cout << "Move disk " << diskToMove << " from " << source << " to " << destination << endl;
        moveDisk( diskToMove-1, spare, destination, source, level+1);
    }
}
```

So how does this work? The above code has been compacted to facilitate using it in a trace below, where we've removed the portions that prepend extra spaces to reflect the recursion level.

```
void moveDisk( int diskToMove, char src, char dest, char spare) {
    if( disk == 1 )
        cout << "Move " << diskToMove << " from " << src << " to " << dest << endl;
    else {
        moveDisk( diskToMove-1, src, spare, dest);
        cout << "Move " << diskToMove << " from " << src << " to " << dest << endl;
        moveDisk( diskToMove-1, spare, dest, src);
    }
}
```

Let's use the recursion trace approach of making multiple copies of the code, although when doing something like this on paper we would usually only write down the values, which are shown below.

```
1) void moveDisk( int diskToMove, char src, char dest, char spare) {
2)     // Move diskToMove from the source tower to the destination tower
3)     if( diskToMove == 1 ) {
4)         printSpace( level);
5)         cout << "Move disk " << diskToMove << " from " << src << " to " << dest << endl;
6)     } else {
7)         moveDisk( diskToMove-1, src, spare, dest);
8)         printSpace( level);
9)         cout << "Move disk " << diskToMove << " from " << src << " to " << dest << endl;
10)        moveDisk( diskToMove-1, spare, dest, src);
11)     }
12) }
```

First let's see the trace of moveDisk( 2, 'A', 'B', 'C')

The disks are numbered as:
```
1) Move diskToMove from the source tower to the destination tower
2) Recursive base case
3) Recursive case
```

---

Let's use the recursion trace approach of making multiple copies of the code, although when doing something like this on paper we would usually only write down the values, which are shown below.

```
void moveDisk( int diskToMove, char src, char dest, char spare) {
    if( disk == 1 )
        cout << "Move " << diskToMove << " from " << src << " to " << dest << endl;
    else {
        moveDisk( diskToMove-1, src, spare, dest);
        cout << "Move " << diskToMove << " from " << src << " to " << dest << endl;
        moveDisk( diskToMove-1, spare, dest, src);
    }
}
```

---

For the three disk example above, the disks are numbered as:

```
1) Move diskToMove from the source tower to the destination tower
2) Recursive base case
3) Recursive case
```

The disks are numbered as:

```
1) Move diskToMove from the source tower to the destination tower
2) Recursive base case
3) Recursive case
```

---

Let's use the recursion trace approach of making multiple copies of the code, although when doing something like this on paper we would usually only write down the values, which are shown below.

```
void moveDisk( int diskToMove, char src, char dest, char spare) {
    if( disk == 1 )
        cout << "Move " << diskToMove << " from " << src << " to " << dest << endl;
    else {
        moveDisk( diskToMove-1, src, spare, dest);
        cout << "Move " << diskToMove << " from " << src << " to " << dest << endl;
        moveDisk( diskToMove-1, spare, dest, src);
    }
}
```

---

For the three disk example above, the disks are numbered as:

```
1) Move diskToMove from the source tower to the destination tower
2) Recursive base case
3) Recursive case
```

The disks are numbered as:
```
void moveDisk( int disk, char src, char dest, char spare) {  
    if (disk == 1) 
        cout <<"Move "<< disk << " from "<< src << " to " <<dest << endl; // 1.
    else { 
        moveDisk( disk-1, src, spare, dest);  
        cout <<"Move "<< disk << " from "<< src << " to " <<dest << endl; // 2.
        moveDisk( disk-1, spare, dest, src);  
    } 
}

Now lets go a step further and see the trace of moveDisk( 3, 'A', 'B', 'C') :

1. void moveDisk( int disk, char src, char dest, char spare) {  
    if (disk == 1) 
        cout <<"Move "<< disk << " from "<< src << " to " <<dest << endl; // 1.
    else { 
        moveDisk( disk-1, src, spare, dest);  
        cout <<"Move "<< disk << " from "<< src << " to " <<dest << endl; // 2.
        moveDisk( disk-1, spare, dest, src);  
    } 
}

2. void moveDisk( int disk, char src, char dest, char spare) {  
    if (disk == 1) 
        cout <<"Move "<< disk << " from "<< src << " to " <<dest << endl; // 3.
    else { 
        moveDisk( disk-1, src, spare, dest);  
        cout <<"Move "<< disk << " from "<< src << " to " <<dest << endl; // 4.
        moveDisk( disk-1, spare, dest, src);  
    } 
}

3. void moveDisk( int disk, char src, char dest, char spare) {  
    if (disk == 1) 
        cout <<"Move "<< disk << " from "<< src << " to " <<dest << endl; // 5.
    else { 
        moveDisk( disk-1, src, spare, dest);  
        cout <<"Move "<< disk << " from "<< src << " to " <<dest << endl; // 6.
        moveDisk( disk-1, spare, dest, src);  
    } 
}

4. void moveDisk( int disk, char src, char dest, char spare) {  
    if (disk == 1) 
        cout <<"Move "<< disk << " from "<< src << " to " <<dest << endl; // 7.
    else { 
        moveDisk( disk-1, src, spare, dest);  
        cout <<"Move "<< disk << " from "<< src << " to " <<dest << endl; // 8.
        moveDisk( disk-1, spare, dest, src);  
    } 
}
void moveDisk(int disk, char src, char dest, char spare) {
    if (disk == 1) {
        // 6.
        cout << "Move " << disk << " from " << src << " to " << dest << endl; // 6.
    }
    else {
        // 7.
        moveDisk(disk-1, src, spare, dest); // 7.
        cout << "Move " << disk << " from " << src << " to " << dest << endl; // 7.
        moveDisk(disk-1, spare, dest, src); // 7.
    }
}

Now back to the original question: How long will it take for the monks to finish moving the 52 golden disks?

Clickers:
1. An alternate version of the story has monks in Hanoi, with 64 disks. How much of a difference does the extra 12 disks make?
   A. 52 / 12 = 4.3333 times longer
   B. Roughly twice as long
   C. 12 times longer
   D. $2^{12} = 4,096$ times longer

Next lets make some timing assumptions, and then see how long it will take.

2. How many moves can be made by highly motivated monks, in a second?
   A. 0.5
   B. 1
   C. 2
   D. 3
   E. 4+

3. For the sake of argument, lets assume that somehow the monks can make 3 moves per second. Which of the numbers below is closest to the number of years it would take the monks to solve the 64 disk version?
   A. 1
   B. 10
   C. 100
   D. 1,000
   E. 100,000

[ What do we need to know to truly answer this question? ]

Consider the 1953 Arthur C. Clark story of the "Nine Billion Names of God" (which you can find online):

Plot Summary:
In a Tibetan lamasery, the monks seek to list all of the names of God. They believe the Universe was created for this purpose, and that once this naming is completed, God will bring the Universe to an end. Three centuries ago, the monks created an alphabet in which they calculated they could encode all the possible names of God, numbering about 9,000,000,000 ("nine billion") and each having no more than nine characters. Writing the names out by hand, as they had been doing, even after eliminating various nonsense combinations, would take another 15,000 years; the monks wish to use modern technology to finish this task more quickly. They rent a computer capable of printing all the possible permutations, and they hire two Westerners to install and program the machine.

Consider creating a new version of the Towers of Hanoi program that counts how many moves are made, and displays how many seconds it takes. Similar to what we need to do for our current programming assignment, we can use a timer to get real-time results, and we can add a variable to count how many moves are made.

(See TowersOfHanoiCounts.cpp on the course web site Class Notes / CodeDoneInClass)

Let's adopt the Arthur C. Clark idea here and use a computer to make the Towers of Hanoi moves.

Clickers:
1. How long do you think it will take if we display the moves, say for 11 disks?
   A) <1 second   B) ~1 second   C) 2-4 seconds   D) 5-10 seconds   E) 11+ seconds

2. How about for 12 disks?
3. What if we just get the computer to "make" the moves, but not print them out? How long will that take for 12 disks?
   A) <1 second  B) ~1 second  C) 2-4 seconds  D) 5-10 seconds  E) 11+ seconds

4. Same question (no printing), but now how long for 64 disks?
   A) <1 second  B) ~1 second  C) 2-4 seconds  D) 5-10 seconds  E) 11+ seconds

Finally we can more accurately answer the question: How long will it take to move 64 disks?

C structs

When programming in C (as opposed to C++) you don't have classes. In order to cluster together data of different types you instead use a struct.

See the separate notes document listed under today's date for examples of structs, again using Date and Employee information, similar to what we saw previously using C++ classes.

In C++ it turns out structs are virtually identical to classes, except the default permissions for data members in structs is public, whereas in classes it is private.