

## Recursion Review

### *Concept of Recursion*

Recursion is a problem-solving technique as well as a way to define an infinite set. For example: to define a list of items we could use the following:

- a. No item and one item are lists
- b. An item followed by a list is a list.
- c. Nothing else is a list of items.

It occurs in environments other than computer science as we will see with the Sleepy Story.

The basic idea in problem solving using recursion is to use the result(s) of one or more simpler problem(s) to solve a larger one, exemplified by:

```
void RecurseFunction ( problem)
{
    if Simple (Problem)                // termination condition
        solve problem directly ( without recursion)
    else
        breakdown problem into subproblem 1, subproblem 2, etc.
        RecurseFunction (subproblem 1);
        RecurseFunction (subproblem 2); // only if more than one
                                         // subproblem has been identified
        construct problem solution from the solutions of the subproblem(s)
}
```

### *General Rules*

1. Must have at least a base case, i.e., a case which does not involve recursion. It could include more than one such case.
2. Each recursive call must get us closer to the base case(s).
3. Beware of the “compound interest rule”
4. Assume that the recursive call work and does what it is supposed to do.

**Example:** Compute the sum of the squares of integers in the range m to n

```
intsumOfSquares (int m, int n)
{
    if (there is only one integer in the range, i.e. m = n)
        the solution is square of m
    else      // there is more than one integer in the range m to n
        add the square of m to the sum of the squares in the range m+1 to n
}
```

We could have gone down as well (added the square of n to the sum of the squares in the range m to n-1).

As mentioned above it can be illustrated in a non-computing environment by the Sleepy Story (**Handout # 1**)

1. Before the recursive call: forward direction
2. After the recursive call: backward direction
3. Parameters: This is how to communicate things that change
4. If we rewrite the Sleepy Story as a program:

```
void tellSleepyStory (baby)
{
    Say "Once upon a time..."
    if sleepy (baby)
        The little baby did not hear. Baby was fast asleep.
        return
    else
        Say "there was a baby.
        It was time to go to sleep, but baby was not sleepy.
        Well, may be baby was a tiny bit sleepy.
        Baby said to mother, "Tell me a story"
        So his mother called tellSleepyStory (next baby)
    return
}
```

### ***Another Example:*** the Tower of Hanoi

As described in Wikipedia, The **Tower of Hanoi** (also called the **Tower of Brahma** or **Lucas' Tower**,<sup>[1]</sup> and sometimes pluralized) is a [mathematical game](#) or [puzzle](#). It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a [conical](#) shape.

The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.

Let us write a program to solve this puzzle