ECE 131 – Programming Fundamentals

Dr. Daryl O. Lee

University of New Mexico

Electrical & Computer Engineering

Recursion

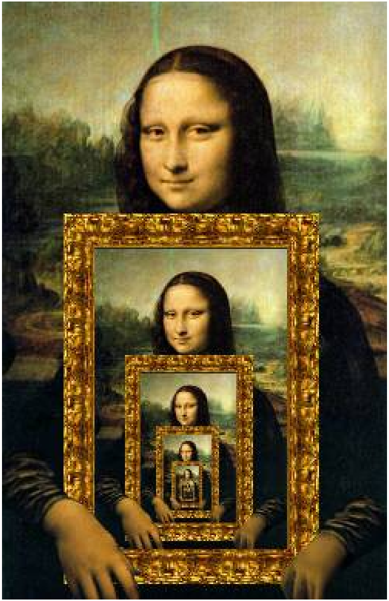
Think about what would happen if a C function called itself?

- Does the program crash, does your computer blow up, does this lead to a cascading sequence of failures the ends in the destruction of the world?
- This is actually an example of recursion — the notion that something is defined with respect to itself.
- Recursion is actually commonly applied in mathematics, language, the arts and computing.
- If you've ever been in a hall-of-mirrors at an amusement park, you've experience recursion.
- We will see that recursion can be very useful as a problem-solving tool in computing.

THE UNIVERSITY of NEW MEXICO

Recursion - A Hall of Mirrors

Recursion in the Arts



Recursion in Mathematics

- Consider the factorial function $f(n) = n! = n \cdot (n-1) \cdots 1$, where n is a natural number.
- Mathematicians commonly use a recurrence relation to express the factorial function as follows:

$$f(n) = \begin{cases} 1, & \text{if } n = 1 \\ n \cdot f(n-1), & \text{if } n > 1 \end{cases}$$

  where n is assumed to be a natural number.
- A recurrence relation is an equation that defines a sequence recursively, i.e., it is defined in terms of itself ($f(n)$ is defined using $f(n-1)$).
- This recurrence relation above states that if $n = 1$, then the value of the function is 1, and if $n > 1$, then the value of the function is computed using the same function at a smaller value of n.
- $f(1)$ is initial condition for this recurrence relation. It's where the recursion "bottoms-out" and therefore stops.

THE UNIVERSITY of NEW MEXICO

Recursion in Computing

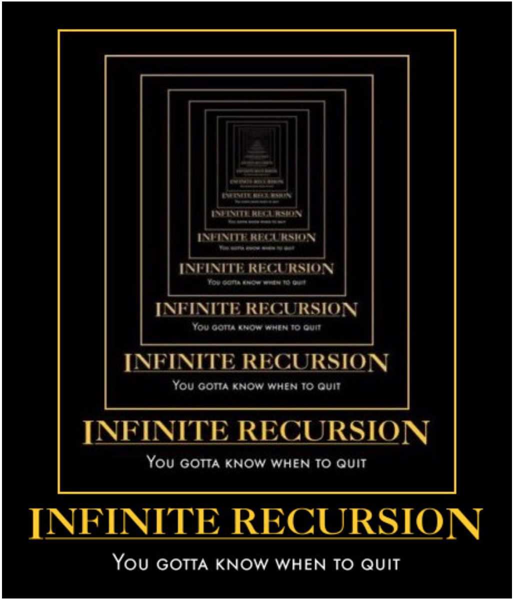
- In computing, when a function calls itself, we call it a recursive function call.
- Recursion is an important tool used in solving computational problems.
- Here's a recursive implementation of the factorial function in C:

```
int factorial(int n) {
    if (n == 1)
        return 1;
    else
        return n * factorial(n-1);  // recursive call
}
```

Consider the call `factorial(4)`, and let us "unfold" the recursion:

activation record	*line*	
		`factorial(4)`
1	4	`  return 4 * factorial(3)`
2	4	`    return 3 * factorial(2)`
3	4	`      return 2 * factorial(1)`
4	2	`        return 1`
3	4	`      return 2 * 1`
2	4	`    return 3 * 2`
1	4	`  return 4 * 6`

Recursion in Computing



Recursion in Computing

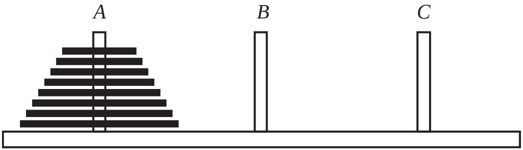
- Implicit in the previous analysis was the assumption that parameters were passed to the `factorial()` function by value. Indeed, this must be the case if the function is to work correctly.
- Consider what would happen if instead, parameters were passed by reference.
  - The parameter value passed to each recursive call of the `factorial()` procedure would be the same.
  - The function as a whole would never reach the base case.
  - The function would not terminate, and thus could not compute the correct value.
- For this reason, pass-by-value is necessary to implement recursive functions.

THE UNIVERSITY of NEW MEXICO

Recursion in Problem Solving

Recursion can be a powerful tool for use in problem solving.

Ex. Towers of Hanoi:



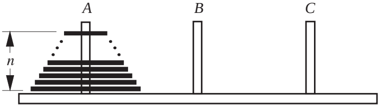
Objective: Move all n disks from peg A to peg C

Rules:

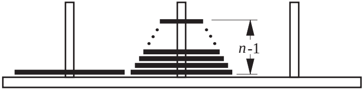
- Can only move one disk at a time.
- Cannot put a larger disk on top of a smaller one.

Towers of Hanoi–Solution

Solution:

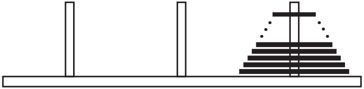


(a)



(b)

(c)



(d)

Towers of Hanoi–Algorithm

Tower(**positive integer** n, **peg** i, **peg** j, **peg** k)

$\triangleright$ move the top n disks on peg i to peg k using peg j

if $n! = 0$ **then**

    Tower($n-1$, i, k, j)

    move top disk on peg i to peg k

    Tower($n-1$, j, i, k)

Towers of Hanoi–Recurrence

- The running time of the previous algorithm is:

$$T(n) = \begin{cases} 1, & \text{if } n = 1 \\ 2T(n-1) + 1, & \text{if } n > 1 \end{cases}$$

- It can be shown that $T(n) = 2^n - 1$.
- What is $T(64)$? 1.8447×10^{19}.
- If it takes 1 sec. to move each disk, how much time would it take to solve this problem assuming there are 64 disks? 5.8494×10^{9} centuries.