Musicomputation: Recursion

Marilyn C. Cole
Courant Institute, NYU
mcole@nyu.edu
Outline

• The concept of *recursion*
  – How does it work?
  – What steps do we take to think recursively?

• The dragon story

• Solving the *Towers of Hanoi* computationally

• An introduction to *Joshephus' problem*
Previously

• Levels of *abstraction*:

  Variables
  • If an expression is a name we created (by declaring a variable) then it represents that variable's corresponding value

  Functions
  • If an expression has a name and takes some amount of parameters, then it represents that function's result or output

• Parts of a *function*:
  • Definition
  • Parameters
  • Return Value
A Simple Abstraction Example

```javascript
function square(x) {
    return x * x;
}

square(4);

x → 4

4 * 4 → 16
```
The Factorial Function: n!

- $n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$

- First note:
  - $n! = n \times (n-1)!$ if $n > 1$

This works well with the “black box” or “wishful thinking” step of recursion: we assume that there is a way to solve the problem for versions of a smaller size.
The Factorial Function: \( n! \)

- \( n! = n \times (n-1)! \)
  - If we have a way to solve the factorial function for \( n-1 \), we must simply multiply \( n \) times that solution: this step is **converting the smaller solution**.

```c
int fact(n)
{
    if (n != 1)
        return n * fact(n-1);
    ...
}
```
The Factorial Function: \( n! \)

- The third step is identifying the base case – realizing when there is no smaller solution – and when to stop!

```c
int fact(n) {
    if (n != 1)
        return n * fact(n-1);
    else
        return 1;
}
```
The Factorial Function: \( n! \)

\[
\text{factorial}(5) \\
5 \times \text{factorial}(4) \\
5 \times (4 \times \text{factorial}(3)) \\
5 \times (4 \times (3 \times \text{factorial}(2))) \\
5 \times (4 \times (3 \times (2 \times \text{factorial}(1)))) \\
5 \times (4 \times (3 \times (2 \times 1))) \\
5 \times (4 \times (3 \times (2))) \\
5 \times (4 \times (6)) \\
5 \times (24) \\
120
\]
“This is the story of how Martin, an alchemist's apprentice, discovered recursion by outsmarting a lazy dragon.”

Puff the Magic Dragon dreams of eating peanut butter sandwiches – and doesn't like when it gets stuck to the roof of his mouth.
The Towers of Hanoi

- Only one disc may be moved at a time

- No disc may be placed on a smaller disc
Let's try it on the board!

• Consider the three steps:
  – Envision what our **black box** does. Can we assume the problem is solvable for a smaller number of discs? Where do we go from there?
  – What must we do to **convert the smaller solution for** \( n-1 \) into a solution for \( n \) discs?
  – Make sure to remember that your algorithm must stop at the **base case**! There is one disc that has no smaller solution.
Josephus Problem

• A theoretical problem in computer science and mathematics

There are \( n \) people standing in a circle to be executed. Going around the circle, \( k \)-1 people are skipped, and the \( k \)th man is executed. The process repeats: \( k \)-1 are skipped, the \( k \)th man is killed. The elimination proceeds (the circle becoming smaller as people are removed), until only one man remains, who is given freedom.

• Let's consider a recursive approach to decide which person is safe – for any \( n \) and \( k \) – though we should note that \( n \) changes over the course of the problem, while \( k \) should stay the same.