CS3L:
Introduction to Symbolic Programming

Lecture 8:
Recursion

Summer 2008

Colleen Lewis
colleenL@berkeley.edu

Announcements

- Midterm grades are posted on UCWISE
- I need to leave 5 minutes early today – Sorry!

Today

- Why we need Recursion
- Factorial
- Define `count` with recursion

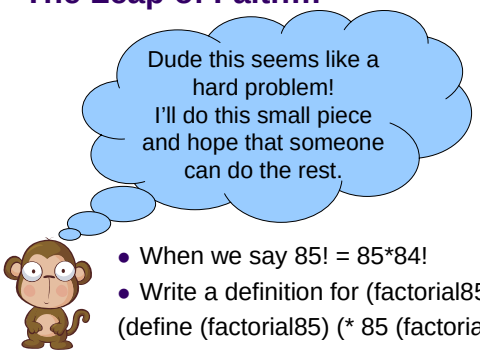
Recursion

- We want to be able to work with arbitrary length sentences
  - Add up a sentence of numbers
  - Find every odd number in a list
  - Find the number of times a word appears in a sentence

Factorial

- 10! = 10*9*8*7*6*5*4*3*2*1
- 10! = 10*9!
- x! = x*(x-1)! (recursive case)
- 1! = 1 (base case)
- 0! = 1 (base case)
- (define (factorial0) 1)
- (define (factorial1) (* 1 (factorial0)))
- (define (factorial2) (* 2 (factorial1)))
- (define (factorial3) (* 3 (factorial 2)))

The Leap of Faith…



- When we say 85! = 85*84!
- Write a definition for (factorial85).

(define (factorial85) (* 85 (factorial84)))

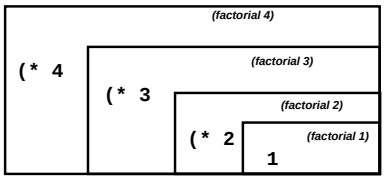
Factorial REAL Recursion

- 10! = 10*9! (recursive case) x! = x*(x-1)!
- 1! = 1 (base case)
- 0! = 1 (base case)

```
(define (factorial x)
  (if (< x 2)
     1
     (* x (factorial (- x 1)))))
```

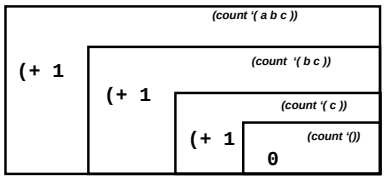
What is (factorial 4)?

```
(define (factorial x)
  (if (< x 2)
     1
     (* x (factorial (- x 1)))))
```



Count the number of words in a sentence

```
(define (count sent)

   (if (empty? sent)  ;no more?

      0                    ;base case: return 0

      (+ 1
        (count (bf sent))) ;recurse on the
                           ; rest of sent

))

>(count '(a b c))
```



All Recursive Procedures Need

1. Base Case (s)
   - Where the problem is simple enough to be solved directly
2. Recursive Cases (s)
   1. **Divide the Problem (Make the problem Smaller!)**
      - into one or more smaller problems
   2. **Invoke the function**
      - Have it call itself recursively on each smaller part
   3. **Combine the solutions**
      - Combine each subpart into a solution for the whole