CS61A Lecture 2

2011-06-21

Colleen Lewis

"Computer Science"

Not really about computers!

Not really a science!

Hierarchy of Abstraction

- Application Programs
- High-level language (Scheme)
- Low-level language (C)
- Machine language
- Architecture (registers, memory, arithmetic unit)
- Circuit elements (gates)
- Transistors
- Solid-state physics
- Quantum mechanics

Functions

- Any number of arguments
- One return value
- Composition of functions like *f(g(x))* from algebra
- Allows easy reordering

REVIEW: Two Types of ('s so far

```
(first '(hello))
```

Call a function

Indicate a sentence

REVIEW: Two Types of ''s so far

```
'hi        '(hello)
```

This is a word

This is a sentence

REVIEW: Way to define variables

```
(define variable value)
```

- Keyword & special form
- Shouldn't be an expression
- An expression

REVIEW: IF & COND Statements

```
(if   <predicate>
      <true case>
      <false case>)
(cond
      (<predicate1>  <return_expression1>)
      (<predicate2>   <return_expression2>)
      (else          <return_expression3>))
```

Example COND Statements

Grouping Parens

```
(define (buzz n)
 (cond
      ((equal? (remainder n 7) 0)   'buzz)
      ((member? 7 n)                'buzz)
      (else                         n)))
```

Administrative

- Click on links on the class webpage
  - Lots of resources
  - inst.eecs.berkeley.edu/~cs61a
- Sign-up for Piazzza
- Make sure you get Scheme working at home
- Make use of tutor hours
- READ THE BOOK!
- There will be teamwork in the class (read the general info doc)

Evaluation Order

```
STk>(define (square x) (* x x))
square
STk>(square 3)
```

(square 3)

(* 3 3)

9

Evaluation Order

```
STk> (define (square x) (* x x))
STk> (square (+ 2 3))
```

Applicative Order	*Normal* Order
(square (+ 2 3))	(square (+ 2 3))
(square 5)	(* (+ 2 3) (+ 2 3))
(* 5 5)	(* 5 5)
25	25

Evaluation Order (Solution)

```
STk>(define (zero x) (- x x))
; assume (random 10) returns 1 then 2
```

Applicative Order	*Normal* Order
(zero (random 10))	(zero (random 10))
(zero ***1***)	(-(random 10)(random 10))
(- 1 1)	(- ***1 2***)
0	-1

Try It! Evaluation Order

```
STk>(define (f a b) (+ (g a) b))
STk>(define (g x) (* 3 x))
```

Applicative Order	*Normal* Order
(f (+ 2 3) (- 15 6))	(f (+ 2 3) (- 15 6))

Try It?

- Does Scheme use?

A) Normal Order?
B) Applicative Order?

Special Forms: `if`, `cond`, `define`

```
(if   <predicate>
      <true case>
      <false case>)
```

We don't just use applicative order to evaluate!

```
(cond
      (<predicate1>  <return_expression1>)
      (<predicate2>  <return_expression2>)
      (else          <return_expression3>))
```

Recursion

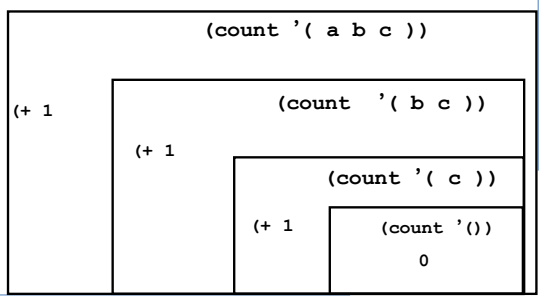
Super important in CS61A

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

Count the number of words in a sentence

```
STk>(count '(a b c))
3
```



Try It!

- Write `count` that takes in a `sentence` and counts the words in the `sentence`.

```
STk> (count '(a b c))
3
```

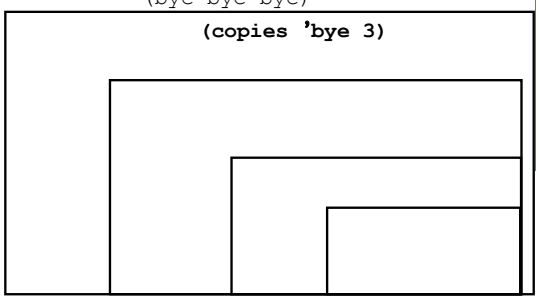
Try It!

- Write `copies` that takes in a `word` and a variable `n` and repeats the word `n` times in a `sentence`.

```
STk> (copies 'hi 2)
(hi hi)
Stk> (copies 'bye 3)
(bye bye bye)
```

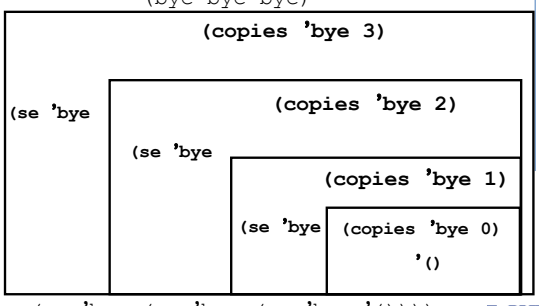
Try it! Count the number of words in a sentence

```
STk>(copies 'bye 3)
(bye bye bye)
```



Count the number of words in a sentence

```
STk>(copies 'bye 3)
(bye bye bye)
```



(se 'bye (se 'bye (se 'bye '())))

Pascal's Triangle

R5C3 = R4C2 + R4C3

	C:0	C:1	C:2	C:3	C:4	C:5
R: 0	1					
R: 1	1	1				
R: 2	1	2	1			
R: 3	1	3	3	1		
R: 4	1	4	6	4	1	
R: 5	1	5	10	10	5	1
R: 6	1	6	15	20	15	6	1

Recursion (Cont.)

R5C3 = R4C2 + R4C3

(R,C) = (R-1,C-1) + (R-1, C)

```
(define (pascal row col)
  (cond
    ((= col 0) 1)
    ((= col row) 1)
    (else (+ 
```

pascal (Cont.) (R,C) = (R-1,C-1) + (R-1, C)

Pascal's Triangle

	C:0	C:1	C:2	C:3	C:4	C:5
R: 0	1					
R: 1	1	1				
R: 2	1	2	1			
R: 3	1	3	3	1		
R: 4	1	4	6	4	1	
R: 5	1	5	10	10	5	1
R: 6	1	6	15	20	15	6	1

(p R C) → (p R-1 C-1), (p R-1 C)

We need a different way to keep track of this recursion

(p 3 1) → (p 2 0), (p 2 1)

(p 2 1) → (p 1 0), (p 1 1)

Unix Review

- ls _____
- cd folder1 _____
- cd .. _____
- cd _____
- mkdir folder2 _____
- rm file1 _____
- emacs file1 & _____

EXTRA: Evaluation Order

```
STk>(define (square x) (* x x))
```

(square (square (+ 2 3)))

Applicative Order

(square (square (+ 2 3)))

Normal Order

EXTRA: Evaluation Order (SOLN)

```
STk>(define (square x) (* x x))
```

(square (square (+ 2 3)))
(square (square 5))
(square (* 5 5))
(square 25)
(* 25 25)

Applicative Order

(square (square (+ 2 3)))
(square (* (+ 2 3) (+ 2 3)))
(* (* (+ 2 3) (+ 2 3)) (* (+ 2 3) (+ 2 3)))

Normal Order

Evaluation Order SOLUTION

```
STk>(define (f a b) (+ (g a) b))
STk>(define (g x) (* 3 x))
```

Applicative Order	*Normal* Order
(f (+ 2 3) (- 15 6))	(f (+ 2 3) (- 15 6))
(f 5 9)	(+ (g (+ 2 3)) (-15 6))
(+ (g 5) 9)	(+ (* 3 (+ 2 3)) (-15 6))
(+ (* 3 5) 9)	(+ (* 3 5) (-15 6))

Count the number of words in a sentence (SOLUTION)

```
(define (count sent)
  (if (empty? sent)  ;no more?
      0              ;base case: return 0
     (+ 1
       (count (bf sent))) ;recurse on the
                          ; rest of sent
))
```

copies (solution)

```
(define (copies n wd)
  (if (< n 1)
    '()
     (se wd
         (copies (- n 1) wd))))
```

pascal Solution

R6C4 = R5C3 + R5C4

(R,C) = (R-1,C-1) + (R-1, C)

```
(define (pascal row col)
  (cond
     ((= col 0) 1)
     ((= col row) 1)
     (else (+ (pascal (- row 1) (- col 1))
              (pascal (- row 1) col)))))
```

Unix Review

- ls — List contents of folder
- cd folder1 — Double click on folder
- cd .. — Go UP one folder level
- cd — Go to home/main folder
- mkdir folder2 — Create new folder
- rm file1 — Remove something
- emacs file1 & — Create file in current folder