6.001 Recitation 3:
Substitution Model, Recursion, and Iteration, Oh My!

RI: Gerald Dalley
14 Feb 2007

The Substitution Model

Newtonian physics, quantum mechanics, relativity, and Scheme...

Rules of the substitution model:

1. if **self-evaluating**, (*e.g.* number, string, `#t`, `#f`), just return value.
2. if **name**, replace it with value associated with that name.
3. if **special form** (*e.g.* `if`, `define`, `lambda`), follow the special form's rules for evaluation
   (a) if **lambda**, create a procedure
   (b) ...
4. if **combination** `(`e_0 e_1 `...` e_n`)`,
   (a) Evaluate subexpressions e_i in any order to produce values `(`v_0 v_1 `...` v_n`)`
   (b) If v_0 is **primitive procedure** (*e.g.* `+`), just apply it to v_1 `...` v_n
   (c) if v_0 is **compound procedure** (created by `lambda`):
      i. Substitute v_1 `...` v_n for corresponding parameters in body of procedure, then repeat on body

Hints:

- Do one thing at a time
- Take small steps
- Sometimes we have a choice of what to do next

```
; Evaluate the following expression, one
; step at a time using the substitution model:
(if (= (+ 5 2) 7) (* (+ 2 3) 5) (/ 4 (- 7 5)))
```

Recursive versus Iterative

From Webster:

- *recur:* to come up again for consideration
- *recursion:* something that calls itself
- *iterate:* repeating, each time getting closer to the desired result

What makes a procedure recursive?

What makes a process recursive?

Designing Recursive Algorithms

- wishful thinking
- decompose the problem
- identify the simple/smallest parts (modular programming)

```
(define (odd? x)
  (not (even? x)))

(define (even? x)
  (not (odd? x)))

; fix even? ...
(define (even? x)
```

- Enumerate assumptions
- Test/predicate for base case (termination of recursive unwinding)
- Recursive call

```
(define (fact n)
   (* n (fact (- n 1))))

; Fix fact, and list the input assumptions you make...
```

High-Level Questions

What is "recursive recursion?"

What is "iterative recursion?"

Why do we like recursive procedures?

Why do some programming guides discourage writing recursive programs?

Recursive → Iterative Recipe

1. What can we use as our partial answer
   - can we "accumulate" something?
2. How do we keep track of what's left to do
   - some sort of counter (count up, count down?)
3. How do we update these variables
   - the partial answer & counter
4. What's the base case?
5. Write out a table
   - fixed size (# of variables) because there are no pending operations
6. Almost always we will write a helper procedure that is the recursive part because we have extra variables to keep track of!

`biggie-size` Returns!

We encode an order as multi-digit number where each digit represents a combo. For example, the order 327 represents a Triple, Double, and biggie-sized Triple. (biggie-size = regular+4)

```
(define order-price
  (lambda (order)
    (if (= order 0)
        0
        (+ (combo-price (remainder order 10))
           (order-price (quotient order 10))))))
```

Is this recursive or iterative?

Rewrite as the other type:

`begin` Special Form

`(begin expr1 expr2 ...)`: The expressions are evaluated sequentially from left to right, and the value of the last expression is returned. This expression type is used to sequence side effects such as input and output.

- allows multiple statements
- scoping
- implicit `begin` inside `define`, `cond`, `lambda`, ...

Consider:

```
(define (foo a)
  (begin
    (define b 5)
    (define (foo-helper x) (+ a x))
    (foo-helper b)))
```

To what do the following expressions evaluate?

`foo`	→		`foo-helper`	→	
`a`	→		`b`	→	
`(foo 3)`	→		`foo-helper`	→	
`a`	→		`b`	→	

Order of Evaluation

Consider the following code:

```
(define (our-display x)
  (display x) ; returns something wierd
  x) ; we return something sensible instead

(define (count1 x)
  (cond ((= x 0) 0)
        (else (our-display x)
              (count1 (- x 1)))))
(define (count2 x)
  (cond ((= x 0) 0)
        (else (count2 (- x 1))
              (our-display x))))

; What does the following generate?
(count1 5)


; What does the following generate?
(count2 4)
```

Evaluate using the substitution model:

```
(count1 5)
```

```
(count2 5)
```