MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.090—Building Programming Experience
IAP 2006

Problem Set 4

1. Desugar the following expressions:

```
(define (foo x)
  (+ x 5))

(let ((x 1))
  x)

(let ((foo (= x 1))
      (bar 7))
  (if foo
      bar
      #f))

(define (weird x y z)     ; this one's odd
  (lambda (foo)
    (+ x y z foo)))
```

2. Evaluate the following expressions, then check with DrScheme

```
(define x 5)

(define (y) (+ 7 7))

(let ((x 3))
  (+ x x))

(let ((x (y))
      (y 7))
  (if (> x 3)
      7
      y)

(let ((mit 12))
  (let ((is (+ mit 1)))
    (let ((hard (- is 7)))
      (+ mit is hard))))
```

3. The following are two different implementations of slow-add, a procedure that adds two numbers which using any arithmetic procedures other than inc and dec:

```
(define (slow-add1 a b)
  (if (= a 0)
      b
      (inc (slow-add1 (dec a) b))))

(define (slow-add2 a b)
  (if (= a 0)
      b
      (slow-add2 (dec a) (inc b))))
```

   For each procedure, indicate whether it gives rise to a recursive or iterative process, and why. Then test it with the stepper to verify your hypothesis. You should define inc and dec as in HW 3. If you cannot access the stepper, change your language setting to "Intermediate student with lambda".

4. Here is the transformation of fact from a recursive to an iterative process that we did in class:

```
; recursive fact
(define (fact n)
  (if (= n 0)
      1
      (* n (fact (- n 1)))))

; iterative fact
(define (fact n)
  (fact-helper n 1))

; helper for iterative version
(define (fact-helper n answer)
  (if (= n 0)
      answer
      (fact-helper (- n 1) (* n answer))))
```

   We also wrote quotient in class:

```
(define (quotient x y)
  (if (< x y)
      0
      (+ 1 (quotient (- x y) y))))
```

   Rewrite quotient to give rise to an iterative process by following the pattern we used for fact. Test your resulting procedure to make sure it works like the original. Then verify that it is iterative by using the Stepper.

5. On a separate piece of paper, draw the box-and-pointer for the given expressions.

```
(cons (cons 1 nil) (cons 2 nil))
```

```
(list (list (list 1) 2) 3)
```

```
(cons nil nil)
```

```
(append (list 3 2) (cons 1 nil))
```

6. Write expression whose values print out like the following:

```
(7)
```

```
("this" "is" "yummy")
```

```
((()))
```

```
(("apples" 3) ("oranges" 2))
```