

6.001 Tutorial 2 – Solutions

David Ziegler

September 20, 2004

1 Substitution Model

The substitution model is a simple way of thinking about how computation works in Scheme. As usual, it's a bit of a lie, but we'll correct the lie shortly. The substitution model tells us what happens when a combination is evaluated:

- Evaluate subexpressions (in any order)
- If the first subexpression is a primitive (built-in) procedure, just apply it to the other values
- Otherwise, substitute the *values* of each subexpression for the parameters in the body of the procedure, and then evaluate the body

```
;; Returns the number of digits in n
(define (num-digits n)
  (if (< n 10)
      1
      (+ 1 (num-digits (/ n 10)))))

(num-digits 2038)
(if (< 2038 10) 1 ...)
(+ 1 (num-digits 203.8))
(+ 1 (+ 1 (num-digits 20.38)))
(+ 1 (+ 1 (+ 1 (num-digits 2.038))))
(+ 1 (+ 1 (+ 1 (+ 1))))
```

2 Recursive/Iterative Processes

We've seen recursive *procedures*, functions that call themselves (possibly indirectly). All functions that are self-referential are recursive. A *process* can be either iterative or recursive.

Iterative processes are those where the value of the recursive call is directly returned – no operations are performed after the call. If the function is named `foo`, the call will be `(foo ...)` all by itself.

Recursive processes are those where the value from the recursive call is operated on before returning – these *pending operations* are what make the process recursive. If the function is named `foo`, the call might be `(- (foo ...) ...)`.

It's often straightforward to convert a recursive process into an iterative process:

```
(define (sum-r a b)
  (if (= a 0)
      b
      (+ 1 (sum-r (- a 1) b))))

(define (sum-i a b)
  (if (= a 0)
      b
      (sum-i (- a 1) (+ b 1))))
```

3 Problems

```
;; From last time:
;; Finds the remainder of x/y using
;; repeated subtraction
(define (remainder x y)
  (if (< x y)
      x
      (remainder (- x y) y)))
```

Iterative or recursive? *Iterative*

```
;; Finds the quotient of x/y
(define (quotient x y)
  (if (< x y)
      0
      (+ 1 (quotient (- x y) y))))
```

Iterative or recursive? *Recursive*

```

;; Alternate implementation
(define (quotient x y)
  (define (helper x y count)
    (if (< x y)
        count
        (helper (- x y) y (+ count 1))))
  (helper x y 0))

```

Iterative or recursive? *Iterative*

```

;; Predicate -- is x divisible by y?
(define (divides? x y)
  (= (remainder x y) 0))

```

Iterative or recursive? *Iterative*

```

;; Predicate -- is p a prime number?
(define (prime? p)
  (define (helper n p)
    (cond ((> (* n n) p) #t)
          ((divides? p n) #f)
          (else (helper (+ n 1) p))))
  (helper 2 p))

```

Iterative or recursive? *Iterative*

```

;; Prints out the prime
;; factorization of n
(define (factorize n)
  (define (helper x n)
    (cond ((> x n)
          #t)
          ((and (divides? n x)
                (prime? x))
           (display x)
           (newline)
           (helper x (/ n x)))
          (else
           (helper (+ x 1) n))))
  (helper 2 n))

```