Quiz 1 Solutions

Problem 1

8
7
9
3
error (wrong # of arguments)

Problem 2

Special forms are needed in Scheme due to the fact that it uses applicative order evaluation. Special forms create special rules for evaluation. For example, with an if statement, the special form will only evaluate the consequent if the predicate is true and will only evaluate the alternative if the predicate is false.

Any four special forms would do. The full list of 15: and, begin, case, cond, define, do, if, lambda, let, let*, letrec, or, quasiquote, quote and set!

Tail recursion is required by the Scheme language definition. With tail recursion, if the procedure is calling itself with no deferred operations, nothing is pushed on the stack. In C, the stack is used for recursive calls, unless you are using a compiler optimization (it is there as an optimization, not as a language standard).

Problem 3

(cdr (car a)) OR (cadr a)
(car (cdr (cdr b))) OR (caddr b)

Problem 4

a. 2 additions
b. 4 additions
c. Scheme uses applicative order evaluation.

Problem 5

(define (car p) (p (lambda (a b) a)))
**Problem 6**

There were two ways to write the procedure: as a recursive process and as an iterative process.

**Recursive process:**

```
(define (compute-f n)
  (if (= n 0)
    1
    (* 2 (compute-f (- n 1)))))
```

**Iterative process:**

```
(define (compute-f n)
  (define (iter x ans)
    (if (= x 0)
      ans
      (iter (- x 1) (* 2 ans)))))
  (iter n 1))
```

\[ T(n) = \Theta(n) \]

\[ S(n) = \Theta(1) \]

\( n \) is dependent upon the value passed to \( compute-f \)

Recursive or iterative process (depending upon your code)

**Problem 7**

```
(define (prod-of-reciprocals a b)
  (product (lambda (x) (/ 1 x))
    a
    (lambda (y) (+ y 1))
    b))

(define (product term a next b)
  (define (prod-iter a ans)
    (if (> a b)
      ans
      (prod-iter (next a) (* ans (term a)))))
  (prod-iter a 1))
```

**Extra Credit**

Every expression has a value (except for errors, infinite loops and the define special form).