1 Chapter 1: Building Abstractions with Procedures

1.1

Below is a sequence of expressions. What is the result printed by the interpreter in response to each expression? Assume that the sequence is to be evaluated in the order in which it is presented.

```
10
10

(+ 5 3 4)
12

(- 9 1)
8

(/ 6 2)
3

(+ (* 2 4) (- 4 6))
6

(define a 3)
a

(define b (+ a 1))
b

(+ a b (* a b))
19

(= a b)
#f

(if (and (> b a) (< b (* a b)))
    b
    a)
4

(cond ((= a 4) 6)
      ((= b 4) (+ 6 7 a))
      (else 25))
16

(+ 2 (if (> b a) b a))
6

(* (cond ((> a b) a)
         ((< a b) b)
         (else -1))
   (+ a 1))
16
```

1.2

Translate the following expression into prefix form:

$$\frac{5+4+(2-(3-(6+\frac{4}{5})))}{3(6-2)(2-7)}$$

```
(/ (+ 5 4 (- 2 (- 3 (+ 6 (/ 4 5)))))
   (* 3 (- 6 2) (- 2 7)))
```

1.3

Define a procedure that takes three numbers as arguments and returns the sum of the squares of the two larger numbers.

```
(define (f x y z)
    (define (sqr a) (* a a))
    (cond ((and (>= x y) (>= y z)) (+ (sqr x) (sqr y)))
          ((and (> x y) (>= z y)) (+ (sqr x) (sqr z)))
          (else (+ (sqr y) (sqr z)))))
```

1.4

Observe that our model of evaluation allows for combinations whose operators are compound expressions. Use this observation to describe the behavior of the following procedure:

```
(define (a-plus-abs-b a b)
    ((if (> b 0) + -) a b))
```

The procedure sums *a* with the absolute value of *b*.

1.5

Ben Bitdiddle has invented a test to determine whether the interpreter he is faced with is using applicative-order evaluation or normal-order evaluation. He defines the following procedures:

```
(define (p) (p))

(define (test x y)
    (if (= x 0)
    0
    y))
```

Then he evaluates the expression

```
(test 0 (p))
```

What behavior will Ben observe with an interpreter that uses applicative-order evaluation? What behavior will he observe with an interpreter that uses normal-order evaluation?

Applicative-order will recurse into infinity as the condition's alternative, (p), evaluated. Normal order will defer calculation of (p) and instead return the condition's consequent 0 as $x == 0$ is evaluated first.