2.4 Consider the following procedures.

```
(define (our-display x)
    (display x)    ; assume display function doesn't return anything useful
    x)             ; add this line so that the procedure returns something useful

(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))))
```

a. What sequence of numbers is displayed for (count1 4)? (Hint: Write out first several substitution steps.)

4 3 2 1

```
(count1 4)
(our-display 4) (count1 3)          <DISPLAY 4>
(count1 3)
(our-display 3) (count1 2)          <DISPLAY 3>
(count1 2)
(our-display 2) (count1 1)          <DISPLAY 2>
(count1 1)
(our-display 1) (count1 0)          <DISPLAY 1>
(count1 0)
=> 0
```

b. What value is returned for (count1 4)? 0

c. What sequence of numbers is displayed for (count2 4)?

1 2 3 4

```
(count2 4)
(count2 3) (our-display 4)
(count2 2) (our-display 3) (our-display 4)
(count2 1) (our-display 2) (our-display 3) (our-display 4)
(count2 0) (our-display 1) (our-display 2) (our-display 3) (our-display 4)
(our-display 1) (our-display 2) (our-display 3) (our-display 4)      <DISPLAY 1>
(our-display 2) (our-display 3) (our-display 4)                      <DISPLAY 2>
(our-display 3) (our-display 4)                                      <DISPLAY 3>
(our-display 4)                                                      <DISPLAY 4>
=> 4
```

Notice that this example is a bit unusual: The deferred operations are not the result of nesting the recursive call inside another expression as an operand; they result from the recursive call's position in the sequence of expressions to be evaluated.