Homework 4
Programming Languages
CS 320
Stephen Giang RedID: 823184070

(1) Give a Scheme expression for a list containing the values such that the number 7 is the first element, the symbol `a` the second, the number -12 the third, and the symbol ??? the fourth.

```
'(7 a -12 ???)
```

(2) Scheme has a function `symbol?` that takes one argument and returns `#t` or `#f` depending on whether the argument is a symbol or not. Give a Scheme expression that determines whether the first element of a three-element list is a symbol. (The result of the evaluation should be `#t`.)

```
(symbol?  (car '(?  a b c) ) )
```

(3) Suppose that into the Scheme evaluator we enter the definition: `(define dozen 12)` Give the value of the Scheme expression: `(number?  dozen)`

```
#t
```

(4) Give two ways to write a Scheme expression of which the value is a one-element list with the empty list as its element.

```
'( '() )   &   '( () )
```

(5) Give a Scheme definition that will determine the sum of 30, 31, 30, 15 and bind the identifier `days-in-semester` to this sum. Be as concise as possible.

```
(define days-in-semester (+ 30 31 30 15))
```

(6) Evaluate this Scheme statement. (Do this by hand first; then check your answer by running it.)

```
((lambda (x) (+ x x)) 4)
```

```
4
```

(7) Evaluate these Scheme statements. (Do this by hand first; then check your answer by running it.)

```
(define reverse-subtract
 (lambda (x y)
   (- y x)))
(reverse-subtract 7 10)
```

```
3
```

(8) These expressions form repetitions. Trace by hand the following loop and show the set of two lists it produces (in other words, the result will be lists within a list).

```
(let loop
     ((numbers '(3 -2 1 6 -5))
      (nonneg '())
      (neg '()))
  (cond ((null?  numbers)
         (list nonneg neg))
        ((>= (car numbers) 0)
         (loop (cdr numbers)
               (cons (car numbers) nonneg)
               neg))
        (else
          loop (cdr numbers)
               nonneg
                (cons (car numbers) neg)))))
```

((6 1 3) (-5 -2))

(9) Here is a function called "deleteall" that strips out all occurrences of a value in a given list. The call to the function: (deleteall 0 '(4 0 3 0 2 0 1)) results in the list (4 3 2 1).

```
(define (deleteall atm list)
   (cond
      ( (NULL? list) '() )
      ( (EQ? atm (CAR list)) (deleteall atm (CDR list)))
      ( ELSE (CONS (CAR list) (deleteall atm (CDR list))))
))
```

Write a recursive Scheme procedure called "tally" that counts the number of occurrences of a value in a given list. Do not use a counter variable, but let the function result in the desired count. For example, the value of `(tally 0 '(4 0 3 0 2 0 1))` will be 3. The procedure will be very similar to `deleteall`.

```
(define (tally atm list)
   (cond
   ( (null?  list) 0 )
   ( (eq?  atm (car list)) (+ 1 (tally atm (cdr list))) )
   ( else (tally atm (cdr list)))
))
```

(10) The binding construct `let` defines Scheme block structure. In a `let` expression, the initial values are computed before any of the variables become bound. Evaluate these expressions by hand showing the effects of lexical binding in Scheme.

```
A. (let ((x 2) (y 3))
          (* x y))
```

6

```
B. (let ((x 2) (y 3))
       (let ((foo (lambda (z) (+ x y z)))
             (x 7))
  ; call foo
    (foo 4)))
```

9