```
;; CMPSC 461 Project 2 Taylan Unal (tuu2)
;;-----------------------------------------------------------------------

;; #1  higher order function dncall
;; Takes 3 parameters (n f x), returns x when n=0, when n=1 returns f(f(x)), when n=2 returns f(f(f(f(x)))), etc.
;; The function returns the result of calling f(x) 2n times.
;; INPUT: f is a boolean function, l is a list of ints.
;; OUTPUT: List of ints that pass condition in boolean function

(define (addOne x) (+ x 1)) ;;just a sample function, can be lambda.

(define (dncall n f x)
  (if (= n 0)
      x
      (dncall (- n 1) f (f (f x))) ;; else case. starts stacking f(f(x)) for each recursive call determined by n
   )
)

;;-----------------------------------------------------------------------

;; #2 Scheme keep-if
;; define function using case analysis and recursion
;; INPUT: f is a boolean function, l is a list of ints.
;; OUTPUT: List of ints that pass condition in boolean function
(define (keep-if f l)
  (define output '())
  (cond ((null? l) '()) ;; if list is null, return empty list.
    ((f (car l)) ;; apply function on first element of list.
     (cons (car l) (keep-if f (cdr l))));;if car l true, constructs new list of it and recursive call of the rest of the list.

    (else (keep-if f (cdr l))) ;; else case, when car l is not true, simply continue with remainder of list.
  )
)

;;-----------------------------------------------------------------------

;; #3a least_helper function
;;Input: a number 'k', a list of numbers 'x'
;;Output: the minimum element from a list of numbers
(define (least_helper k x) ;; k is a number, x is a list of numbers
  (if (null? x) k ;; if list is empty, return k, since k is least in set of k
      (if (< (car x) k)
          (least_helper (car x) (cdr x))
          (least_helper k (cdr x))
      )
   )
)

;;-----------------------------------------------------------------------

;; #3b least function
;; Uses least_helper to break down a list and find minimum element within list.
;; Input: Inputs a list of numbers
;; Output: the minimum element from a list of numbers
(define (least l)
  (if (null? l)'()
      (least_helper (car l) (cdr l)) ;;starts recursive callstack on list, calls least_helper on rest of list.
  )
)

;;-----------------------------------------------------------------------


```

```
;;-----------------------------------------------------------------------
;; #4 to-words function
;; Input: A number from -99 to 99 inclusive
;; Output: text representation of number in list format.
(define (to-words n)
  (define ones '(one two three four five six seven eight nine ten eleven twelve thirteen fourteen fif
teen sixteen seventeen eighteen nineteen))
  (define tens '(twenty thirty fourty fifty sixty seventy eighty ninety))

  (cond ;; START TOP COND
    ((< n -99) '(ERROR: OUT OF BOUNDS))
    ((> n 99) '(ERROR: OUT OF BOUNDS))
    ((= n 0) '(zero))

    ;;START NEGATIVE READOUT
    ((< -100 n 0)
     (cons 'negative ;;constructs a list based on output of below.
      (cond
        ((< 0 (abs n) 20) ;;ALL VALUES FROM 1-19 are in ONES LIST
         (cons (list-ref ones (- (abs n) 1)) '() ))
        ((< 20 (abs n) 30)
         (cons 'twenty (list (list-ref ones (- (abs n) 21)))))
        ((< 30 (abs n) 40)
         (cons 'thirty (list (list-ref ones (- (abs n) 31)))))
        ((< 40 (abs n) 50)
         (cons 'fourty (list (list-ref ones (- (abs n) 41)))))
        ((< 50 (abs n) 60)
         (cons 'fifty (list (list-ref ones (- (abs n) 51)))))
        ((< 60 (abs n) 70)
         (cons 'sixty (list (list-ref ones (- (abs n) 61)))))
        ((< 70 (abs n) 80)
         (cons 'seventy (list (list-ref ones (- (abs n) 71)))))
        ((< 80 (abs n) 90)
         (cons 'eighty (list (list-ref ones (- (abs n) 81)))))
        ((< 90 (abs n) 100)
         (cons 'ninety (list (list-ref ones (- (abs n) 91)))))
        ((integer? (quotient (abs n) 10)) ;;used for clean tens values without ones
         (list (list-ref tens (- (quotient (abs n) 10) 2))))
              (else '(ERROR: CANT RETURN RESULT (NEG)))))) ;; if no case found for n. UNLIKELY
          ;;END NEGATIVE READOUT

          ;; START POSITIVE READOUT
          ((< 0 n 20) ;;ALL VALUES FROM 1-19 are in ONES LIST
           (cons (list-ref ones (- n 1)) '() ))
          ((< 20 n 30)
           (cons 'twenty (list (list-ref ones (- n 21)))))
          ((< 40 n 50)
           (cons 'fourty (list (list-ref ones (- n 41)))))
          ((< 50 n 60)
           (cons 'fifty (list (list-ref ones (- n 51)))))
          ((< 60 n 70)
           (cons 'sixty (list (list-ref ones (- n 61)))))
          ((< 70 n 80)
           (cons 'seventy (list (list-ref ones (- n 71)))))
          ((< 80 n 90)
           (cons 'eighty (list (list-ref ones (- n 81)))))
          ((< 90 n 100)
           (cons 'ninety (list (list-ref ones (- n 91)))))
          ((integer? (quotient n 10))
           (list (list-ref tens (- (quotient n 10) 2)))) ;;used for clean tens values without ones
          (else '(ERROR: CANT RETURN RESULT (POS))) ;; if no case found for n. UNLIKELY
          ;; END POSITIVE READOUT
        );;; END TOP COND
      )
      ;;-----------------------------------------------------------------------
```

```
;;-----------------------------------------------------------------------
;; #5 Develop scheme program that calculates relevant word counts
;; Member function is a boolean function that checks if an element is member of given list.
;; INPUT: Input a element, list of elements.
;; OUTPUT: Boolean #t or #f if element is a member of a list or not
(define (member? x l)
    (if (null? l) #f
        (if (equal? x (car l)) #t
            (member? x (cdr l)))
    )
)
;;-----------------------------------------------------------------------
;; #5a write a function filterWords that takes a list of words and irrelevant words, returns a
n output list with irrelevant words removed.
;; INPUT: List of words, list of words to filter out
;; OUTPUT: returns list with irrelevant words filtered out

(define (filterWords words irrWords)
  (if (null? words) '()
      (if (member? (car words) irrWords)
          (filterWords (cdr words) irrWords)
          (append (list (car words)) (filterWords (cdr words) irrWords))
      )
   )
)
;;-----------------------------------------------------------------------

;; #5b write a function iniWordCountList that takes a list of words, creates a word-
count list
;; INPUT: A list of words
;; OUTPUT: each word has count of 1. So create a set of sublists within a larger list

(define (iniWordCountList words)
  (map (lambda (x) (list x 1)) words) ;; for every word in words, creates a map of the word an
d 1.
)

;;-----------------------------------------------------------------------

;; #5c write a function mergeWordCounts, takes in two inputs
;; First input is word-count pair and second is word-count list
;; function will generate a new word-count list.

(define (mergeWordCounts a x) ;;  a x. list1 list2
  (if (null? x) (list a)
      (if (equal? (car a) (car (car x)))
          (cons (list (car a) (+ (car (cdr a)) (car (cdr (car x))))) (cdr x))
          (cons (car x) (mergeWordCounts a (cdr x)))
      )
  )
)
;;-----------------------------------------------------------------------
```

```
176.        ;;-----------------------------------------------------------------------
177.
178.        ;; #5d write a function mergeByWord that takes word-count list, outputs updated word-
    count list that condenses
179.        ;; Input: A given word-count list [ie '((time 1) (is 1))']
180.        ;; Output: Reduced list with no duplicates, increases wordcount of each element based on how m
    any duplicates
181.
182.        (define (mergeByWord l) ;; f is part c function, l is input list
183.          (if (null? l) '()
184.            (mergeWordCounts (car l) (mergeByWord (cdr l))) ;;(mergeWordCounts (car l) (mergeWordCount
    s (cdr l)))
185.          )
186.        )
187.
188.        ;;mergeByWord based on Reduce:
189.        (define (reduce f l v)
190.          (if (null? l) v
191.              (f (car l) (reduce f (cdr l) v))))
192.
193.        ;;-----------------------------------------------------------------------
194.
195.        ;; #5e Write a function relevantWord
196.        ;; Input: A list of words, A list of relevant words
197.        ;; Output: Correct word count list
198.
199.        ;; x is input words, y is relevant words
200.        (define (relevantWordCount x y)
201.          (mergeByWord (iniWordCountList (filterWords x y)))
202.        )
203.
204.        ;;-----------------------------------------------------------------------
```