```
//Scheme Assignment

; Scheme function that returns the reverse of its simple list parameter.
;(Q11 from programming assignment chapter 15)

(define (reverseL l)
      (if (null? l)
         '()
         (append (reverseL (cdr l)) (list (car l)))
      )
)

;Scheme function that returns the union of two simple list parameters that
;represent sets. (Q13 from programming assignment chapter 15)

(define (union a b)
      (cond
      ( (null? a) b)
      ( (null? b) a)
      ( (member (car a) b) (union (cdr a) b))
      ( else (union (cdr a) (cons (car a) b)))
      )
)

;Scheme function that takes a simple list of numbers as its parameter and
;returns a list identical to the parameter list except with the numbers in
;ascending  order.  (Q16 from programming assignment chapter 15)

(define (selectionSort L)
            (cond
            ((null? L) '())
            (else (cons (smallest L (car L))
                  (selectionSort ( remove L (smallest L (car L)))))
            )
      )
)

(define (smallest L A)
            (cond
            ((null? L) A)
            ((< (car L) A) (smallest (cdr L) (car L)))
            (else (smallest (cdr L) A))
            )
)

(define (remove L A)
            (cond
            ((null? L) '())
            (( = (car L) A) (cdr L))
            (else (cons (car L) (remove (cdr L) A)))
            )
)
```

```
;Scheme function that takes a simple list of numbers as its parameter and
;returns the largest and smallest numbers in the list. (Q17 from programming
;assignment chapter 15)

(define (maxmin L)
        (cond
        ((null? L) '())
        (else (list (maximum L (car L)) (minimum L (car L))))
        )
)

(define (minimum L A)
        (cond
        ((null? L) A)
        (( < (car L) A) (minimum (cdr L) (car L)))
        (else (minimum (cdr L) A))
        )
)

(define (maximum L A)
        (cond
        ((null? L) A)
        (( > (car L) A) (maximum (cdr L) (car L)))
        (else (maximum (cdr L) A))
        )
)

; Scheme function that takes a simple list as its parameter and returns a
;list of all permutations of the given list. (Q18 from programming assignment
;chapter 15)

(define (permutation s)
        (cond
        ((null? s) '())
        ((null? (cdr s)) (list s))
        (else
         (let splice ( (l '()) (m
                               (car s)) (r (cdr s)))
         (append
         (map (lambda (x) (cons  m x))
         (permutation (append l r)))
         (if (null? r)
             '()
             (splice (cons m l) (car r) (cdr r)))))
        )
    )
  )
```

```
//Haskell Assignment

import Data.List


{- haskell function that returns the reverse of its simple list parameter.
  (Q11 from programming assignment chapter 15)-}

reverseList xs = if null xs
                 then xs
                 else reverseList (tail xs) ++ [head xs]

{- haskell function that returns the union of two simple list parameters that
  represent sets. (Q13 from programming assignment chapter 15)-}

unionL a b
      | null a = b
      | null b = a
      | elemIndex (head a) b /= Nothing = unionL (tail a) b
      | otherwise = unionL (tail a) b ++ [(head a)]

{- haskell function that takes a simple list of numbers as its parameter and
  returns a list identical to the parameter list except with the numbers in
  ascending order.  (Q16 from programming assignment chapter 15 -}

quicksort :: (Ord a) => [a] -> [a]
quicksort [] = []
quicksort (x:xs) =
   let smallerSorted = quicksort [a | a <- xs, a <= x]
       biggerSorted = quicksort [a | a <- xs, a > x]
   in  smallerSorted ++ [x] ++ biggerSorted

{- haskell function that takes a simple list of numbers as its parameter and
  returns the largest and smallest numbers in the list. (Q17 from programming
  assignment chapter 15)-}

imax s a
     | null s = a
     | (>) (head s) a = imax (tail s) (head s)
     | otherwise = imax (tail s) a

imin s a
     | null s = a
     | (<) (head s) a = imin (tail s) (head s)
     | otherwise = imin (tail s) a

maxmin s
     | null s = []
     | otherwise = [imax s (head s)] ++ [imin s (head s)]

{- haskell function that takes a simple list as its parameter and returns a list
  of all permutations of the given list. (Q18 from programming assignment
```

chapter 15)-}

```
permutation :: Eq a => [a] -> [[a]]
permutation [] = [[]]
permutation xs = [x : ys | x <- xs, ys <- permutation (delete x xs)]
```