```
;; keep-rel  (num num -> bool) num  list-of-nums -> list-of-nums
;; Purpose: keep all the numbers  in the input list that have the
;;     relation given by the function argument to the number
;;     argument  (whew!)
(define (keep-rel   relation num alon)
   (local [(define filter-rel  alon)      ;; treat relation & num as invariant
              (cond
                [(empty?  alon)  empty]
                [(cons?   alon)
                     (cond
                        [(relation (first alon) num)
                          (cons (first alon) (filter-rel (rest alon)))]
                        [else (filter-rel (rest alon))] ) ] ))
          ]
          (filter-rel alon) ))

(define (keep-gt-9 alon)
     (keep-rel  >  9 alon))
```

```
;; keep-bet-5-9: list-of-numbers -> list-of-numbers
;; Purpose: returns a list containing those numbers in the
;;           input list whose value is between 5 and 9,
;;           inclusive
(define (keep-bet-5-9 alon)
  (cond
    [(empty? alon)   empty]
    [(cons?    alon)
     (cond
       [(and (>= (first alon) 5) (<= (first alon) 9))
         (cons (first alon) (keep-bet-5-9 (rest alon)))]
       [else  (keep-bet-5-9 (rest alon))]
     ) ] ) )
```

```
;; bet-5-9?: number -> boolean
;; Purpose:  test if the argument is between five and nine,
;;             inclusive
(define (bet-5-9? anum)
    (and (>= num 5) (<= num 9)))

;; keep-bet-5-9: list-of-numbers -> list-of-numbers
;; Purpose: returns a list containing those numbers in the
;;             input list whose value is between 5 and 9,
;;             inclusive
(define (keep-bet-5-9 alon)
   (cond
      [(empty? alon)   empty]
      [(cons?    alon)
       (cond
          [(bet-5-9? (first alon))
            (cons (first alon) (keep-bet-5-9 (rest alon)))]
          [else  (keep-bet-5-9 (rest alon))]
       ) ] ) )
```

```
;; bet? :  num num num -> boolean
;; Purpose: determines if the third argument lies
;;            numerically between the 1st & 2nd arguments
(define (bet?  lower upper  anum)
   (and  (>= num lower) (<= num upper)))

;; keep-bet : num num list-of-numbers -> list-of-numbers
;; Purpose: keeps all the numbers lying between 1st & 2nd
;;            arguments
(define (keep-bet  lower  upper  alon)
   (local
       [(define (filter-bet alon)
           (cond
              [(empty? alon)  empty]
              [(cons?    alon)
                   (cond
                     [(bet? lower upper (first alon))
                       (cons (first alon) (filter-bet (rest alon)))]
                     [else  (filter-bet  (rest alon))])]))]
       (filter-bet alon) ))

(define (keep-bet-5-9 alon)
    (keep-bet 5 9 alon))
```

```
(define (keep  …  alon)
   (local
      [(define (filter  alon)
       (cond
          [(empty? alon) empty)]
          [(cons?    alon)
              (cond
               [( … (first alon))
                    (cons (first alon) (filter (rest alon)))]
               [else (filter (rest alon))] )] ))]
      (filter alon) ))
```

```
(define (keep  keep-elt? alon)
   (local
       [(define (filter  alon)
        (cond
           [(empty? alon) empty)]
           [(cons?    alon)
               (cond
                [(keep-elt? (first alon))
                     (cons (first alon) (filter (rest alon)))]
                [else (filter (rest alon))] )] ))]
       (filter alon) ))


(define (keep-lt-5 alon)
     (local  [(define (lt-5? num) (< num 5))]
        (keep  lt-5?  alon)  ))

(define (keep-bet-5-9 alon)
    (local [(define (bet-5-9?  num) (bet?  5  9 num))]
          (keep  bet-5-9?  alon) ))
```

```
;; keep-fee : list-of-symbol -> list-of-symbol
;; Purpose: return the list containing every occurrence of
;;          the symbol 'fee
(define (keep-fee  alos)
   (local [(define (is-fee?  asym)(=  'fee  asym))]
           (keep   is-fee?  alos) ))
```