```
;; CS 135 :: Fall 2017 :: Posted solution :: A04 :: lists.rkt


;; 2(a)

;;(sum-positive lst) sums the positive numbers in lst
;; sum-positive (listof Int) -> Int
;; Examples:
(check-expect (sum-positive empty) 0)
(check-expect (sum-positive (cons 5 (cons -3 (cons 4 empty)))) 9)

(define (sum-positive lst)
  (cond
    [(empty? lst) 0]
    [(positive? (first lst)) (+ (first lst) (sum-positive (rest lst)))]
    [else (sum-positive (rest lst))]))

;; Test:
(check-expect (sum-positive (cons -3 (cons -5 empty))) 0)


;; 2(b)

;;(contains? elem lst) determines if elem is in lst
;; contains?: Any (listof Any) -> Bool
;; Examples:
(check-expect (contains? 7 empty) false)
(check-expect (contains? 'fun (cons 'racket (cons 'is (cons 'fun empty)))) true)
(check-expect (contains? 0 (cons 1 (cons 2 (cons 3 empty)))) false)

(define (contains? elem lst)
  (cond
    [(empty? lst) false]
    [(equal? elem (first lst)) true]
    [else (contains? elem (rest lst))]))

;; Tests:
(check-expect (contains? 0 empty) false)
(check-expect (contains? (make-posn 0 1)
                         (cons (make-posn 1 1)
                               (cons (make-posn 0 1) empty))) true)
(check-expect (contains? 1 (cons 'a (cons "a" (cons 1 empty)))) true)


;; 2(c)

;;(has-duplicate? lst) determines whether lst has an element that appears
;;    more than once
;; has-duplicate?: (listof Any) -> Bool
;; Examples:
(check-expect (has-duplicate? empty) false)
(check-expect (has-duplicate? (cons 1 (cons 2 (cons 2 empty)))) true)

(define (has-duplicate? lst)
  (cond
    [(empty? lst) false]
    [(contains? (first lst) (rest lst)) true]
    [else (has-duplicate? (rest lst))]))
```

```
;; Tests:
(check-expect (has-duplicate? (cons 1 (cons 2 (cons 3 empty)))) false)
(check-expect (has-duplicate? (cons 1 (cons 2 (cons 1 empty)))) true)


;; 2(d)

;;(keep-ints lst) produces a list of the integers in lst
;; keep-ints: (listof Any) -> (listof Int)
;; Examples:
(check-expect (keep-ints empty) empty)
(check-expect (keep-ints (cons 'a (cons 1 (cons "b" (cons 2 empty)))))
              (cons 1 (cons 2 empty)))

(define (keep-ints lst)
  (cond [(empty? lst) empty]
        [(integer? (first lst)) (cons (first lst) (keep-ints (rest lst)))]
        [else (keep-ints (rest lst))]))

;; Tests:
(check-expect (keep-ints empty) empty)
(check-expect (keep-ints (cons 10.1 (cons 'a empty))) empty)



;; CS 135 :: Fall 2017 :: Posted solution :: A04 :: div-by-3.rkt


;; 3(a)

;; A Nat3 is one of:
;; * 0
;; * 1
;; * 2
;; * (+ Nat3 3)


;; 3(b)

;; nat3-template: Nat3 -> Any
(define (nat3-template n)
  (cond
    [(= n 0) ...]
    [(= n 1) ...]
    [(= n 2) ...]
    [else (... (nat3-template (- n 3)) ...)]))


;; 3(c)

;;(div-by-3? n) determines if n is divisible by 3
;; div-by-3?: Nat3 -> Bool
;; Examples:
(check-expect (div-by-3? 0) true)
(check-expect (div-by-3? 1) false)
(check-expect (div-by-3? 2) false)
(check-expect (div-by-3? 12) true)
```

```
(define (div-by-3? n)
  (cond [(= n 0) true]
        [(= n 1) false]
        [(= n 2) false]
        [else (div-by-3? (- n 3))]))

;; Test:
(check-expect (div-by-3? 13) false)



;; CS 135 :: Fall 2017 :: Posted solution :: A04 :: trains.rkt


(require "a04lib.rkt")


;; 4(a)

;; A Unit-Type is (anyof 'L 'B 'T 'P 'C)

;; A Unit is a (make-unit Unit-Type Nat)

;; A Train is one of:
;; * empty
;; * (cons Unit Train)
;; requires: each serial number is unique


;; 4(b)

;; string->train consumes a string and breaks it into a list of characters.  It
;; then recurses over that list and a list of serial numbers, making a
;; Unit out of each character/number pair.  The pairs are put into a list.


;; 4(c)

;;(headed-by? train type) determines if the first unit in train is a given type
;; headed-by?: Train Unit-Type -> Bool
;; Example:
(check-expect (headed-by? (string->train "LPC") 'L) true)

(define (headed-by? train type)
  (and (cons? train)
       (symbol=? type (unit-type (first train)))))

;; Tests:
(check-expect (headed-by? (string->train "PC") 'L) false)
(check-expect (headed-by? (string->train "") 'L) false)


;; 4(d)

;;(ends-with-caboose? train) determines if train has exactly one caboose
;;    and is its last unit
;; ends-with-caboose?:  Train -> Bool
```

```
;; Examples:
(check-expect (ends-with-caboose? (string->train "")) false)
(check-expect (ends-with-caboose? (string->train "LBTC")) true)

(define (ends-with-caboose? train)
  (cond [(empty? train) false]
        [(symbol=? (unit-type (first train)) 'C) (empty? (rest train))]
        [else (ends-with-caboose? (rest train))]))

;; Tests:
(check-expect (ends-with-caboose? (string->train "LBT")) false)
(check-expect (ends-with-caboose? (string->train "LCBTC")) false)
(check-expect (ends-with-caboose? (string->train "LBTCC")) false)


;; 4(e)

;;(remove-unit train serial) removes the unit with a given serial number
;;   from train (if such a unit exists)
;; remove-unit: Train Nat -> Train
;; Examples:
(check-expect (remove-unit (string->train "") 1) empty)
(check-expect (remove-unit (string->train "LBC") 3)
              (cons (make-unit 'L 2) (cons (make-unit 'C 5) empty)))

(define (remove-unit train serial)
  (cond [(empty? train) train]
        [(= (unit-serial (first train)) serial) (rest train)]
        [else (cons (first train) (remove-unit (rest train) serial))]))

;; Tests:
(check-expect (remove-unit (string->train "LBC") 49) (string->train "LBC"))
(check-expect (remove-unit (string->train "LBC") 2)
              (cons (make-unit 'B 3) (cons (make-unit 'C 5) empty)))
(check-expect (remove-unit (string->train "LBC") 5)
              (cons (make-unit 'L 2) (cons (make-unit 'B 3) empty)))

;; 4(f)

;;(headed-by-car? train) determines if the first unit in train is a box car,
;;    tank car, or passenger car
;; headed-by-car?: Train -> Bool
;; Examples:
(check-expect (headed-by-car? (string->train "BC")) true)
(check-expect (headed-by-car? (string->train "LC")) false)

(define (headed-by-car? train)
  (or (headed-by? train 'B)
      (headed-by? train 'T)
      (headed-by? train 'P)))

;; Tests:
(check-expect (headed-by-car? (string->train "TC")) true)
(check-expect (headed-by-car? (string->train "PC")) true)


;;(proper-train? train) determines if the train is a proper train
;; proper-train?: Train -> Bool
```

```
;; Examples:
(check-expect (proper-train? (string->train "")) true)
(check-expect (proper-train? (string->train "L")) true)
(check-expect (proper-train? (string->train "CCLTBL")) false)

(define (proper-train? train)
  (cond [(empty? train) true]
        [(empty? (rest train)) true]
        [(headed-by? train 'L) (proper-train? (rest train))]
        [(headed-by-car? train)
         (and (not (headed-by? (rest train) 'L))
              (proper-train? (rest train)))]
        [else (and (headed-by? (rest train) 'C)
                   (proper-train? (rest train)))]))

;; Tests:
(check-expect (proper-train? (string->train "LC")) true)
(check-expect (proper-train? (string->train "BL")) false)
(check-expect (proper-train? (string->train "BB")) true)
(check-expect (proper-train? (string->train "CL")) false)
(check-expect (proper-train? (string->train "CC")) true)
(check-expect (proper-train? (string->train "LLLBPTCCC")) true)


;; ************************ Alternate Proper-Train Solution ************************


;;(remove-cabooses train) produces train without its leading cabooses
;; remove-cabooses: Train -> Train
;; Examples:
(check-expect (remove-cabooses (string->train "")) empty)
(check-expect (remove-cabooses (string->train "CCL"))
              (cons (make-unit 'L 5) empty))

(define (remove-cabooses train)
  (cond
    [(empty? train) empty]
    [(headed-by? train 'C) (remove-cabooses (rest train))]
    [else train]))

;; Test:
(check-expect (remove-cabooses (string->train "CCCCC")) empty)


;;(remove-cars train) produces train without its leading cars
;; remove-cars: Train -> Train
;; Examples:
(check-expect (remove-cars (string->train "")) empty)
(check-expect (remove-cars (string->train "BTPL"))
              (cons (make-unit 'L 7) empty))

(define (remove-cars train)
  (cond
    [(empty? train) empty]
    [(headed-by-car? train) (remove-cars (rest train))]
    [else train]))

;; Test:
```

```
(check-expect (remove-cars (string->train "BBTTPP")) empty)


;;(remove-locomotives train) produces train without its leading locomotives
;; remove-locomotives: Train -> Train
;; Examples:
(check-expect (remove-locomotives (string->train "")) empty)
(check-expect (remove-locomotives (string->train "LLC"))
              (cons (make-unit 'C 5) empty))

(define (remove-locomotives train)
  (cond [(empty? train) empty]
        [(headed-by? train 'L) (remove-locomotives (rest train))]
        [else train]))

;; Test:
(check-expect (remove-locomotives (string->train "LLLL")) empty)


;;(proper-train/alt? train) determines if the train is a proper train
;; proper-train/alt?: Train -> Bool
;; Examples:
(check-expect (proper-train/alt? (string->train "")) true)
(check-expect (proper-train/alt? (string->train "CCLTBL")) false)

(define (proper-train/alt? train)
  (empty? (remove-cabooses (remove-cars (remove-locomotives train)))))

;; Tests:
(check-expect (proper-train/alt? (string->train "LLL")) true)
(check-expect (proper-train/alt? (string->train "BPT")) true)
(check-expect (proper-train/alt? (string->train "CCC")) true)
(check-expect (proper-train/alt? (string->train "LLBPT")) true)
(check-expect (proper-train/alt? (string->train "LLCC")) true)
(check-expect (proper-train/alt? (string->train "BPTCC")) true)
(check-expect (proper-train/alt? (string->train "LLBPTCC")) true)
(check-expect (proper-train/alt? (string->train "CCL")) false)
(check-expect (proper-train/alt? (string->train "BPTL")) false)
(check-expect (proper-train/alt? (string->train "CCB")) false)
```