Dada la siguiente función auxiliar:

```
(define (simbolo->char simbolo pos)
   (string-ref (symbol->string simbolo) pos))

(check-equal? (simbolo->char '30 0) #\3)
(check-equal? (simbolo->char 'AC 1) #\C)
```

escribe el cuerpo de la siguiente función que comprueba si dos símbolos tienen la misma letra en la posición indicada, utilizando la función auxiliar anterior:

```
(define (igual? pos simbolo1 simbolo2)
      
)

(check-true (igual? 1 'ABC 'XBC))
(check-false (igual? 2 'ABC 'ABX))
```

Completa el código de la función *proporcion-digitos* considerando que los chequeos *check-equal?* son correctos, y recuerda que *char-numeric?* da cierto para los caracteres que son dígitos numéricos:

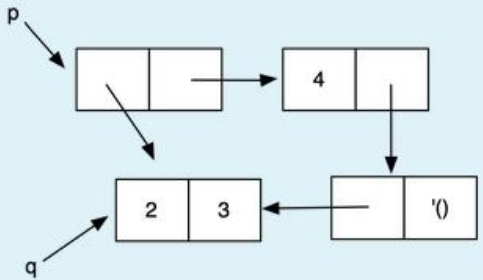
```
(check-equal? (incrementa-izq (cons 5 8)) '(6 . 8))
(check-equal? (incrementa-der (cons 3 6)) '(3 . 7))

(define (proporcion-digitos lista)
    (if (null? lista)
        
        (if (char-numeric? (car lista))
            (incrementa-izq (proporcion-digitos (cdr lista)))
            (incrementa-der (proporcion-digitos (cdr lista))))))

(check-equal? (proporcion-digitos
                (string->list "abc123de4f5g")) '(5 . 7))
(check-equal? (string->list "Hola") '(#\H #\o #\l #\a))
```

Comprobar

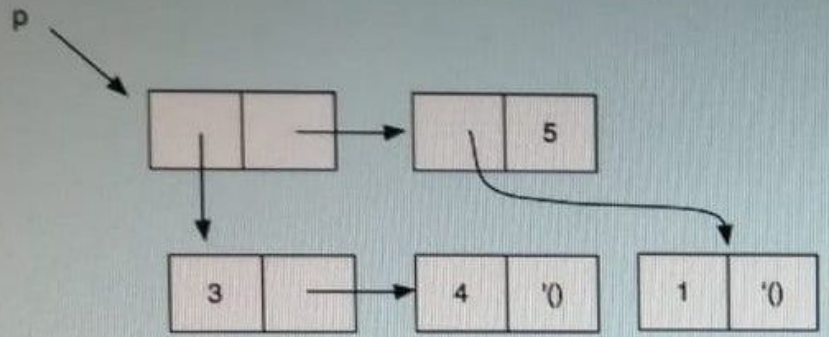
Dado el siguiente box & pointer:



Completa la expresión que lo genera, utilizando el mínimo número de llamadas a list y cons.

```
(define q (cons 2 3))
(define p                                   )
```

Dado el siguiente diagrama box & pointer:



Escribe la expresión más pequeña posible en Scheme que lo construye, usando el mínimo número de llamadas a list y cons:

```
(define p (cons (cons 3 (cons 4 (list))) (cons (cons 1 (list)) 5)))
```

Rellena el hueco de la siguiente función recursiva que recibe una lista de números y devuelve una pareja cuya parte izquierda sea la suma de los elementos de la lista y la parte derecha el producto de dichos elementos:

```
(define (suma-izq-mult-der n pareja)
  (cons (+ n (car pareja))
        (* n (cdr pareja))))

(define (suma-producto lista)
  (if (null? (cdr lista))
      (cons (car lista) (car lista))
      (suma-izq-mult-der (car lista) (suma-producto (cdr lista)))))

(check-equal? (suma-producto '(2 3 4)) '(9 . 24))
```

Completa la función f para que funcione tal y como indican las pruebas. Devuelve una pareja, en su parte izquierda se indica el número de caracteres y en la derecha el número de símbolos que hay en la lista pasada como parámetro.

```
(define (incr-d p)
    (cons (car p) (+ 1 (cdr p))))
(define (incr-i p)
    (cons (+ 1 (car p)) (cdr p)))

(define (f lista)
  (cond
    ((null? lista) (cons 0 0))
    ((char? (car lista))      (incr-i (f (cdr lista)))    )
    ((symbol? (car lista)) ...)
    (else ...)))

(check-equal? (f '(a #\a q w #\s "hola")) (cons 2 3))
(check-equal? (f '(v e)) (cons 0 2))
(check-equal? (f '(#\d)) (cons 1 0))
```

Rellena el hueco de la siguiente función recursiva que comprueba si si en una lista de números todos cumplen que su siguiente número es múltiplo del anterior. Utiliza la función auxiliar:

```
(define (multiplo? a b)
  (= 0 (remainder a b)))

(define (secuencia-multiplos? lista)
  (if (null? (cdr lista))
      #t
      (if (not (multiplo? (cadr lista) (car lista))) #f
          (secuencia-multiplos? (cdr lista)))))


(check-true (secuencia-multiplos? '(2 4 16)))
(check-false (secuencia-multiplos? '(3 7 14)))
```