Pregunta 1

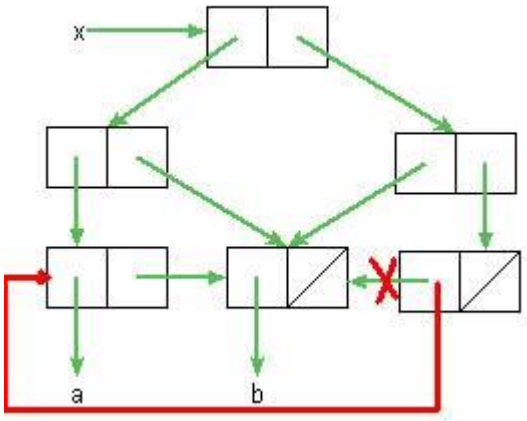
a)

```
(define (suma-polinomios pol1 pol2)
  (cond ((null? pol1) pol2)
        ((null? pol2) pol1)
        (else (cons (+ (car pol1) (car pol2))
                    (suma-polinomios (cdr pol1) (cdr pol2))))))
```

b)

b.1)
```
(define x
        (let* ((y (cons 'b '()))
               (w (cons 'a y))
               (z (cons y '())))
               (cons (cons w y) (cons y z))))
```

b.2)



Pregunta 2

a)

```
(define (split-k lista n)
  (if (< (length lista) n)
      (list lista)
      (append (list (primeros-n lista n))
              (split-k (resto-n lista n) n))))

(define (primeros-n lista n)
  (if (= n 0)
      '()
      (cons (car lista)
            (primeros-n (cdr lista) (- n 1)))))

(define (resto-n lista n)
  (if (= n 0)
      lista
      (resto-n (cdr lista) (- n 1))))
```

b)

```
(define (split-k! lista n)
  (if (< (length lista) n)
      (list lista)
      (let ((lista2 (corta-n lista n)))
        (cons lista
              (split-k! lista2 n)))))

(define (corta-n lista n)
  (if (= n 1)
      (let ((lista2 (cdr lista)))
        (set-cdr! lista '())
        lista2)
      (corta-n (cdr lista) (- n 1))))
```

Pregunta 3

a) Una posible barrera de abstracción sería:

Constructor:
`(make-matriz fil col):` devuelve una matriz de `fil` filas y `col` columnas.

Selectores:
`(num-filas-matriz matriz):` devuelve el número de filas que tiene la matriz `matriz`.
`(num-cols-matriz matriz):` devuelve el número de columnas que tiene la matriz `matriz`.

Funciones específicas:
`(insert-fila-matriz fila matriz)`: añade la fila `fila` a la matriz `matriz`.
`(get-elem-matriz fila col matriz):` devuelve el elemento situado en la fila `fila` y columna `col` de la matriz `matriz`.
`(get-fila-matriz fila matriz):` devuelve la fila indicada por `fila` de la matriz `matriz`.
`(get-col-matriz col matriz):` devuelve la columna indicada por `col` de la matriz `matriz`.

Implementación de dos funciones de esta barrera de abstracción:

```
(define (num-filas-matriz m)
  (if (null? m) 0
  (length m)))

(define (num-cols-matriz m)
  (if (null? m) 0
  (length (car m))))
```

b)

```
(define (make-matrix rows cols start)
  (if (= rows 0) '()
      (cons (make-fila cols start)
            (make-matrix (- rows 1) cols (+ start cols)))))

(define (make-fila cols start)
  (if (= cols 0) '()
      (cons start (make-fila (- cols 1) (+ start 1)))))
```

c)

```
(define (transpuesta m)
  (cond
    ((null? (car m)) '())
    (else (cons (map car m) (transpuesta (map cdr m))))))
```

Pregunta 4

a) Utilizamos una tabla hash similar a la definida en el tema de programación dirigida por los datos, que relaciona los identificadores con los procedimientos. La interfaz de esta tabla hash sería:

`(put identificador procedimiento)`: añade un procedimiento a la tabla hash, asociándolo al identificador
`(get identificador)`: devuelve el procedimiento asociado al identificador

Ejemplos de uso:

```
(put 'suma +)
(put 'doble (lambda (x) (+ x x))
(get 'doble) -> devuelve el procedimiento
```

Implementación de calculadora:

```
(define (calculadora funcion n)
  (let ((proc (get funcion)))
    (if (null? proc)
       (error "funcion desconocida")
       (proc n))))
```

Para añadir nuevas funciones a la calculadora ya no hay que tocar la función `calculadora`, basta con añadir una nueva pareja (identificador, procedimiento) a la tabla hash usando la función `put`:

```
(define (añade-funcion ident proc)
   (put ident proc))
```

b)

```
(define suma +)

(define (media . args)
  (/ (apply suma args) (length args)))

(define (calculadora funcion . args)
  (cond
    ((equal? funcion 'factorial) (factorial (car args)))
    ((equal? funcion 'cuadrado) (cuadrado (car args)))
    ((equal? funcion 'suma) (apply suma args))
    ((equal? funcion 'media (apply media args)))
    (else (error "funcion desconocida"))))
```

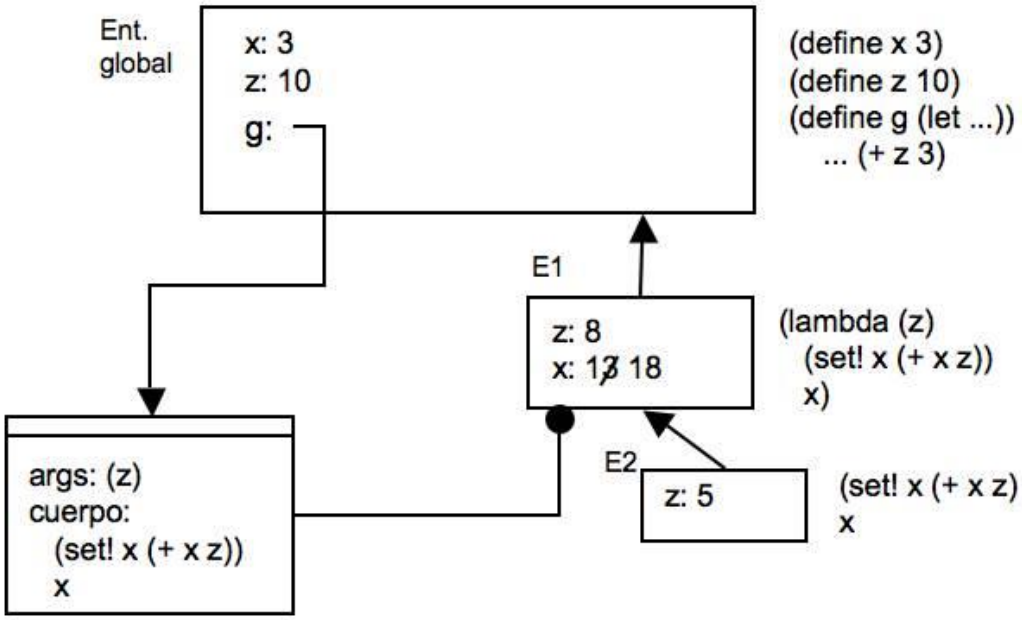
Pregunta 5

```
(define (suma-nivel-tree n tree)
  (if (= n 0) (dato tree)
      (suma-nivel-forest n (hijos tree))))

(define (suma-nivel-forest n forest)
  (if (null? forest) 0
      (+ (suma-nivel-tree (- n 1) (car forest))
         (suma-nivel-forest n (cdr forest)))))
```

Pregunta 6

a)



b) 18
c) `(define z 6)`