1. Evalúa la siguiente expresión usando el tipo de alcance y estrategia de evaluación que se indica. Es necesario incluir el ambiente final en cada caso.

```
(let (a (- 5 -5))
   (let (b (- a a))
      (let (foo (lambda () (- a b)))
         (let (a (+ -3 -3))
            (let (b (+ a a))
               (foo))))))
```

(a) Alcance estático y evaluación glotona.
Solucion. Hagamos el ambiente de evaluación como sigue.

b	-12
a	-6
foo	10
b	0
a	10

Ahora realizaremos la evaluación de la expresión con los siguientes pasos.

1. Evaluar foo.
2. Buscamos foo en la tabla y vemos que vale 10, por lo tanto foo = 10.

(b) Alcance dinámico y evaluación glotona.
Solucion. Hagamos el ambiente de evaluación como sigue.

b	-12
a	-6
foo	10
b	0
a	10

Ahora realizaremos la evaluación de la expresión con los siguientes pasos.

1. Evaluar foo.
2. Buscamos foo en la tabla y vemos que vale 10, por lo tanto foo = 10.

(c) Alcance estático y evaluación perezosa.
Solucion. Hagamos el ambiente de evaluación como sigue.

b	(+ a a)
a	(+ -3 -3)
foo	(lambda () (- a b))
b	(- a a)
a	(- 5 -5)

Ahora realizaremos la evaluación de la expresión con los siguientes pasos.

1. Evaluar foo.
2. Buscamos foo en la tabla y vemos que foo = (lambda () (- a b)).
3. Sustituimos con el valor de a,b de abajo, por lo que (- (- 5 -5) (- a a))
   = (- (- 5 -5) (- (- 5 -5) (- 5 -5)))
   = (- (- 5 -5) (- 10 10)) = (- 10 0) = 10
   entonces foo = 10.

(d) Alcance dinámico y evaluación perezosa.
Solucion. Hagamos el ambiente de evaluación como sigue.

b	(+ a a)
a	(+ -3 -3)
foo	(lambda () (- a b))
b	(- a a)
a	(- 5 -5)

Ahora realizaremos la evaluación de la expresión con los siguientes pasos.

1. Evaluar foo.
2. Buscamos foo en la tabla y vemos que foo = (lambda () (- a b)).
3. Sustituimos con el valor de a,b de arriba, por lo que (- (+ -3 -3) (+ a a))
   = (- (+ -3 -3) (+ (+ -3 -3) (+ -3 -3)))
   = (- (+ -3 -3) (+ -6 -6)) = (- -6 -12) = 6
   entonces foo = 6.

2. Dada la siguiente funcion take:

```
take :: Int -> [a] -> [a]
take 0 _ = []
take n (x:xs) = x:(take (n-1) xs)
```

(a) ¿Cuántos registros de activación genera la llamada take 3 [1,2,3,4,5]?
Solucion. Veamos los siguientes registros de activación.

1 : take(2) [2,3,4,5]
n = 3 [1,2,3,4,5]
take 3 [1,2,3,4,5]

→

2 : take(1) [3,4,5]
n = 2 [2,3,4,5]
take 2 [2,3,4,5]

→

3 : take(0) [4,5]
n = 1 [3,4,5]
take 1 [3,4,5]

→

[ ]
n = 0 [4,5]
take 0 [4,5]

Por lo que tenemos 4 registros de activación generados.

(b) ¿Cuántos registros de activación son usados a la vez con la llamada take 3 [1,2,3,4,5]?
Solucion. De igual forma solamente se genera 4 registros de activación ya que se utilizan 4 en cada ejecución.

(c) Optimiza la función anterior usando la técnica de recursión de cola.
Solucion. Primero creamos la función prima como sigue.

take' :: Int → [a] → [a] → [a]
take' 0 _ acc = acc
take' n (x:xs) acc = take'(n-1)xs(x:acc)

Por lo que
take :: Int → [a] → [a]
take n (x:xs) = take' n (x:xs) [ ]

(d) ¿Cuántos registros de activación genera la llamada de cola take 3 [1,2,3,4,5]?
Solucion. Veamos los siguientes registros de activación.

take' 3 [1,2,3,4,5] 1
n = 3 [1,2,3,4,5]
take 3 [1,2,3,4,5]

Por lo que tenemos 1 registro de activación generado ya que no se vuelve a ocupar la función take.

(e) ¿Cuántos registros de activación son usados a la vez con la llamada de cola take 3 [1,2,3,4,5]?
Solucion. De igual forma solamente se genera un registro de activación ya que se utiliza uno en cada ejecución.