Lenguajes de Programación
Practica 2

Karla Ramírez Pulido José Eliseo Ortiz Montaño

Semestre 2024-1
Facultad de Ciencias UNAM

Fecha de inicio: 28 de agosto 2023
Fecha de entrega: 11 de septiembre 2023

Instrucciones

La realización y entrega de la práctica deberá realizarse en equipos de a lo más 3 personas*. La entrega debe respetar el orden de las funciones que es especificado en este documento. El uso de funciones auxiliares está totalmente permitido, sin embargo, deben ser declaradas justo después de la función principal que hace una de ellas.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Ejercicio 1
;; [Documentación de la función]
(define (my-fun a b c d) ... )
;; [Documentación de la función auxiliar - incluir una breve descripción de
;; la motivación de la función]
(define (an-aux-fun lst1 lst2) ... )
...
...
...
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Las funciones deben incluir comentarios, tanto de documentación como del proceso de desarrollo. Estos deben ser claros, concisos y descriptivos.

Queda estrictamente prohibido utilizar funciones primitivas del lenguaje que resuelvan directamente los ejercicios.

Se deberá subir la versión final de su práctica al apartado del classroom correspondiente antes de la fecha límite. Esto solo debe realizarlo un integrante del equipo; el resto deberá marcar como entregada la actividad y en un comentario privado especificar quienes son los miembros del equipo.

El archivo debe ser llamado `practica2.rkt` y en forma de documentación, deberán incluir los datos de los miembros del equipo.

*Cualquier situación con respecto a este punto será tratada de acuerdo a las particularidades del caso. Para esto, acercarse al ayudante del rubro del laboratorio a la brevedad.

Ejercicios

1. Considerando la siguiente definición del tipo de dato abstracto, el cual representara un punto en un plano y resuelve los siguientes ejercicios.

```
(define-type Punto
    [punto (x number?) (y number?)])
```

   a) Define la función `punto-medio`, la cual recibe dos puntos, `p` y `q`, y calcula el punto medio ente estos, si alguno de los dos no es un punto regresa un error. La fórmula para calcular el punto medio es $p_{medio} = (\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2})$.

```
;; punto-medio :: Punto Punto ->Punto
(define (punto-medio p q) ... )

>(punto-medio (punto 2 2) (punto 2 8))
(punto 2 5)
```

   b) Define la función `distancia`, la cual recibe dos puntos, `p` y `q`, y calcula la distancia entre ellos, si alguno de los dos no es un punto regresa un error. La fórmula para calcular la distancia entre dos puntos es: $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

```
;; distancia :: Punto Punto ->number
(define (distancia p q) ... )

>(distancia (punto 2 2) (punto 2 8))
6
```

2. Considera la siguiente definición del tipo de dato abstracto, el cual representara una lista y resuelve los siguientes ejercicios.

```
(define-type Lista
    [Vacia]
    [Cons (cabeza any?) (resto Lista ?)])
```

   a) Define la función `longitud`, la cual recibe una lista y obtiene el numero de elementos que esta misma contiene.

```
;; longitud :: Lista ->number
(define (longitud ls) ... )

>(longitud (Cons 1 (Cons 2 (Cons 3 (Cons 4 (Vacia))))))
4
```

   b) Define el predicado `pertence?`, la cual recibe un elemento y una lista y devuelve `#t` si dicho elemento se encuentra en la lista y `#f` en cualquier otro caso.

```
;; pertenece? :: any Lista ->boolean
(define (pertenece? e ls) ... )

>(pertenece? 3 (Cons 1 (Cons 2 (Cons 3 (Cons 4 (Vacia))))))
#t
```

c) Define la función `intercala`, la cual recibe dos listas e intercala los elementos de la segunda con los de la primera, si alguna de las dos listas no contiene elementos se regresa la otra.

```
;; intercala :: Lista Lista ->Lista
(define (intercala ls ks) ... )

>(intercala (Cons 1 (Cons 2 (Cons 3 (Cons 4 (Cons 5 (Cons 6 (Vacia)))))))
            (Cons 6 (Cons 7 (Vacia))))
(Cons 1 (Cons 6 (Cons 2 (Cons 7 (Cons 3 (Cons 4 (Cons 5 (Cons 6 (Vacia)))))))))
```

d) Define la función `aplana`, la cual recibe una lista cuyos elementos pueden ser otras listas y devuelve una lista que contenga todos los elementos de la lista, así como de las sublistas.

```
;; aplana :: Lista ->Lista
(define (aplana ls) ... )

>(aplana (Cons (Cons 1 (Cons 2 (Cons 3 (Vacia)))) (Cons 4 (Cons 5 (Vacia)))))
(Cons 1 (Cons 2 (Cons 3 (Cons 4 (Cons 5 (Vacia))))))
```

3. Considera la siguiente definición del tipo de dato abstracto, el cual representara un árbol binario de búsqueda, donde todos los elementos del subárbol derecho son menores o iguales a la raíz del árbol y todos los elementos del subárbol izquierdo son estrictamente mayores a esta, y resuelve los siguientes ejercicios.

```
1 (define−type ArbolBinarioDeBusqueda
2     [ArbolVacio]
3     [ABB (elemento number?) (izq ArbolBinarioDeBusqueda?) (der ArbolBinarioDeBusqueda?)])
```

a) Define la función `elimina` que toma un árbol binario de búsqueda y un elemento, elimina este último del árbol y regresa el árbol resultante de dicha operación.

```
;; elimina :: ArbolBinarioDeBusqueda any ->ArbolBinarioDeBusqueda
(define (elimina ar e) ... )

>(elimina (ABB 5 (ABB 6 (ABB 7 (ArbolVacio) (ArbolVacio)) (ArbolVacio))
          (ABB 4 (ArbolVacio) (ArbolVacio))) 5)
(ABB 6 (ABB 7 (ArbolVacio) (ArbolVacio)) (ABB 4 (ArbolVacio) (ArbolVacio)))
```

b) Define la función `mapea-arbol` que recibe un árbol binario de búsqueda y una función, aplica esta última a cada elemento del árbol y devuelve el árbol resultado de esto.

```
;; mapea-arbol :: ArbolBinarioDeBusqueda procedure ->ArbolBinarioDeBusqueda
(define (mapea-arbol ar f) ... )

>(mapea-arbol (ABB 5 (ABB 6 (ABB 7 (ArbolVacio) (ArbolVacio)) (ArbolVacio))
          (ABB 4 (ArbolVacio) (ArbolVacio))) add1)
(ABB 6 (ABB 7 (ABB 8 (ArbolVacio) (ArbolVacio)) (ArbolVacio))
       (ABB 5 (ArbolVacio) (ArbolVacio)))
```

c) Define la función `hojas` que recibe un árbol binario de búsqueda y regresa una lista con los valores de cada una de sus hojas.

```
;; hojas :: ArbolBinarioDeBusqueda ->(listof any)
(define (hojas ar) ... )

>(hojas (ABB 5 (ABB 6 (ABB 7 (ArbolVacio) (ArbolVacio)) (ArbolVacio))
          (ABB 4 (ArbolVacio) (ArbolVacio))))
'(7 4)
```

4. **Punto extra.** Define la función `mas-repetido`, la cual recibe una lista y devuelve el elemento con mayor número de repeticiones en esta. Si hay dos o más elementos repetidos el mismo número de veces, regresa el primero en aparecer dentro de la lista y si la lista está vacía lanza un error.

```
;; mas-repetido :: (listof any) ->any
(define (mas-repetido ls) ... )
```

```
>(mas-repetido (list 1 2 3 3 6 6 7 7))
3
```