Lenguajes de Programación

Ejercicios propuestos

Karla Ramírez Pulido Manuel Soto Romero Alejandro Hernández Mora
Silvia Díaz Gómez Pedro Ulises Cervantes González

Semestre 2021-2
Facultad de Ciencias, UNAM

1. Ejercicios popuestos

1. Un procedimiento **recursivo** que calcule a^b.

```
;; Procedimiento que eleva el numero a, a la potencia b
;; potencia: number number
(define (potencia a b) ...)

> (potencia 2 -3)
1/8
```

2. Un procedimiento que reciba un número natural n y devuelva la lista ordenada de menor a mayor, de los números menores o iguales a n, desde el cero.

```
;; menores: number -> (listof number)
(define (menores n) ... )

> (menores 13)
'(0 1 2 3 4 5 6 7 8 9 10 11 12 13)
```

3. Un procedimiento **recursivo** que reciba un número natural n y calcule la suma de los primeros n números naturales al cuadrado. En este ejercicio no se permite usar la fórmula conocida para esto.

```
;; suma-cuadradosR: number -> number
(define (suma-cuadradosR n) ...)

> (suma-cuadrados 10)
385
```

4. Un procedimiento que recibe un número natural y devuelve el día de la semana correspondiente a n. Empezando por el número cero, al cuál le corresponde el día *lunes*

```
;; Procedimiento que determina el día de la semana correspondiente
;; diaSemana: number -> string
(define (daSemana n) ... )

> (diaSemana 0)
``Lunes''
```

```
> (diaSemana 9)
‘‘Miércoles’’
```

5. Una predicado que reciba un elemento a, una lista l y decida si a pertenece a l.

```
;; pertenece?: a (listof a) -> boolean
(define (pertenece? a l) ...)
```

6. Una función que reciba una lista l con elementos y devuelva una lista sin repeticiones con los elementos de l.

```
;; eliminaRep: (listof a) -> (listof a)
(define (eliminaRepetidos lista) ...)
```

7. Una función que reciba una lista como parámetro y nos devuelva una lista con el orden de sus elementos invertidos.

```
;; Funcion que nos da una lista invertida de la lista pasada como parametro
;; reversa-lista: (listof a) -> (listof a)
(define (reversa-lista lista) ... )

> (reversa-lista (list 1 2 3 4))
’(4 3 2 1)
```

8. Un predicado que verifique si una lista (no necesariamente homogénea[1]) es un palíndromo.

```
;; palindromo-lista?: (listof a) -> Boolean
(define (palindromo? lista) ... )

> (palindromo? ’(1 "hola" 1))
#t
```

9. Una función que calcule el valor numérico más grande que contenga una lista l.

```
;; Funcion que nos da el elemento maximo de una lista
;; maximo: (listof a) -> number
(define (maximo lista) ... )

> (maximo ’(-10 3 4 1 0.0 7.99 -5))
7.99
```

[1] Es decir que puede tener elementos de diferentes tipos.

10. Una función que calcule una lista con los divisores de n.

```
;; Funcion que nos da el una lista de divisores de un numero pasado como
   parametro
;; divisores: number -> (listof number)
(define (divisores n) ... )

> (divisores 65)
'(1 5 13 65)
```

Considera la siguiente definición del tipo de abstracto de datos para crear puntos en el plano. Servirá para modelar figuras geométricas.

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

11. Una función que reciba dos puntos $p = (x_1, y_1)$ y $q = (x_2, y_2)$; y calcule el punto medio entre p y q. Si alguno de los dos argumentos 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 -> number
(define (punto-medio p q) ... )

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

12. Una función que reciba dos puntos $p = (x_1, y_1)$ y $q = (x_2, y_2)$; y calcule la distancia entre p y q. Si alguno de los dos argumentos no es un punto, arroja un error. El cálculo de la distancia entre dos puntos está dado por la fórmula: $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

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

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

13. Define un tipo abstracto de datos para crear figuras geométricas. Considera un constructor para:

    - Círculos: Recibe como parámetro un punto, que representa el centro del círculo y un número mayor que cero, que representa el radio.
    - Triángulos: Recibe como parámetro los tres puntos del triángulo.
    - Cuadrados: Recibe como parámetro un punto, que representa el de la esquina superior izquierda del cuadrado, y un número positivo que representa la longitud del lado del cuadrado.
    - Rectángulos: Recibe como parámetro un punto, que representa el de la esquina superior izquierda del rectángulo y dos números positivos que representan el largo y la altura de la base del

rectángulo, respectivamente[2].

14. Una función que reciba una lista l como parámetro y devuelva el elemento con mayor número de repeticiones en la lista l. Si hay dos o más elementos repetidos el mismo número de veces, regresa el primero de éstos en aparecer de izquierda a derecha en la lista original. Si la lista está vacía, lanza un error.

```
;; masRepetido (listof a) -> a
;; (define (masRepetido lista) ... )

> (masRepetido (list 1 2 3 3 6 6 7 7))
3
> (masRepetido (list 1 2 3 4))
1
> (masRepetido (list 1 2 3 4 5 6 3))
3
```

15. La función `conjunto-cuadrado` que recibe una lista y devuelve el producto cruz de los elementos de dicha lista consigo misma. Considera que la lista podría tener elementos repetidos, en cuyo caso se deben de quitar las repeticiones o ignorarlas al calcular el resultado.

```
;; Función que recibe una lista y devuelve el producto cruz de los
;; elementos de dicha lista, consigo misma.
;; conjunto-cuadrado:(listof number) -> (listof (pairof number))
(define (conjunto-cuadrado lista) ... )
> (conjunto-cuadrado (list 1 2 3 3 4))
'((1 1) (1 2) (1 3) (1 4) (2 1) (2 2) (2 3) (2 4) (3 1) (3 2) (3 3) (3 4) (4 1)
   (4 2) (4 3) (4 4))
```

16. La función `cambio` que recibe como parámetro un total de dinero en efectivo a pagar, un monto con el que se pagó. La función calcula el cambio que se debe devolver en las denominaciones de: $\$50, \$20, \$10, \$5, \$2, \$1$.

```
;; Función que calcula el cambio que tenemos que devovler según el
;; monto a cobrar y el monto pagado. Devuelve la cantidad de monedas de las
;; denominaciones $50, $20, $10, $5, $2, $1.
;; area-cono: number number -> (number number number number number number)
(define (cambio total pago) ... )
> (cambio 124 200)
(1 1 0 1 0 1)
```

[2]Para todas las figuras geométricas, los parámetros deben recibirse en ese orden y debes respetar el nombre de los constructores.

17. La función `descomposicion-primos` que recibe un número y calcule su descomposición de números primos.

```
;; Función que calcula la descomposición en factores primos de un número
;; descomposicion-primos: number -> (listof (pairof number))
(define (descomposicion-primos n) ... )
> (descomposicion-primos 14175)
'((3 4) (5 2) (7 1))
```

18. Define un tipo abstracto de datos para crear un árbol binario de búsqueda, no tiene que ser balanceado. Este árbol tiene como propiedad que todos los descendientes izquierdos de cada nodo, contienen un elemento estrictamente menor que la raíz. Por otro lado, los descendientes que contienen un elemento mayor o igual a la raíz, están a la derecha de ésta.
Considera los constructores para:

- El árbol vacío: No recibe parámetros
- Una hoja: Recibe como parámetro un elemento de tipo number.
- Un nodo: Recibe como parámetro un elemento de tipo number, dos árboles binarios de búsqueda, que representan hijo `izquierdo` y `derecho`

Los parámetros deben recibirse en ese orden y debes respetar el nombre de los constructores.

```
(define-type ABB
  [vacio]
  [hoja ...]
  [nodo ...])
```

19. La función `agrega` que recibe un número n, un árbol de búsqueda binario y devuelve el árbol después de agregar n.

```
;; Función que recibe un número, un árbol binario y agrega el elemento al
;; árbol de búsqueda binario.
;; agrega: number ABB -> number
(define (agrega n arbol) ... )
(agrega 6(agrega 4 (agrega 5 (agrega 3 vacio))))
> (nodo 3 (vacio) (nodo 5 (hoja 4) (hoja 6)))
```

20. La función `contiene` que evalua si un árbol de búsqueda binario contiene un elemento.

```
;; Función que recibe un árbol binario, un elemento y
;; devuelve verdadero si el elemento está contenido en el árbol,
;; falso en otro caso.
;; contiene: ABB -> number -> boolean
(define (contiene arbol e) ... )
(contiene (agrega 6(agrega 4 (agrega 5 (agrega 3 vacio)))) 8)
> #f
```