Práctica 2: Programación funcional en Scheme

Ejercicio 1

En clase de teoría hemos visto que el símbolo `cond` es una **forma especial**. Vamos a crear una función `new-cond` que tome como argumentos 2 booleanos (`expr1` y `expr2`) y 3 valores a devolver (`then1` , `then2` y `else`):

```
(define (new-cond expr1 then1 expr2 then2 else)
    (cond
        (expr1 then1)
        (expr2 then2)
        (else else)))
```

En principio, parece que la función `new-cond` es equivalente a la forma especial `cond` . Por ejemplo, el resultado de evaluar las siguientes expresiones es el mismo:

```
(cond ((= 2 1) (+ 1 1)) ((> 3 2) (+ 2 3)) (else (- 10 3)))
(new-cond (= 2 1) (+ 1 1) (> 3 2) (+ 2 3) (- 10 3))
```

Sin embargo, no siempre pasa esto. Por ejemplo, en las siguientes expresiones:

```
(cond ((= 2 2) 1) ((> 3 2) 2) (else (/ 3 0)))
(new-cond (= 2 2) 1 (> 3 2) 2 (/ 3 0))
```

a) Explica detalladamente por qué `cond` y `new-cond` funcionan de forma distinta.

b) Comprueba ahora el funcionamiento de las primitivas `and` y `or` . ¿Son formas especiales o funciones? ¿Por qué? Pon ejemplos y explica tu respuesta detalladamente.

Ejercicio 2

Escribe el predicado `(maximo lista)` que reciba una lista numérica como argumento y devuelva el mayor número de la lista. Suponemos listas de 1 o más elementos. No puedes utilizar la función `max` de Scheme. Puedes definir y utilizar una función auxiliar `mayor` .

La **formulación recursiva** del caso general podemos expresarla de la siguiente forma:

```
;;
;; Formulación recursiva de (maximo lista):
;;
;; El máximo de los elementos de una lista es el mayor entre
;; el primer elemento de la lista y el máximo del resto de la lista
;;
```

Ejemplos:

```
(maximo '(2 3 4 5 6 7 8)) ⇒ 8
(maximo '(2 3 4 8 5 6 7)) ⇒ 8
```

Ejercicio 3

a) Diseña la función recursiva `(ordenada? lista-nums)` que recibe como argumento una lista de números. Devuelve `#t` si los números de la lista están ordenados de forma creciente y `#f` en caso contrario. Suponemos listas de 1 o más elementos.

Escribe primero la formulación recursiva del caso general, y después realiza la implementación en Scheme.

Ejemplos:

```
(ordenada? '(-1 23 45 59 100)) ⇒ #t
(ordenada? '(12 45 -1 293 1000)) ⇒ #f
(ordenada? '(3)) ⇒ #t
```

b) Diseña la función recursiva `(ordenada-palabra? pal)` que recibe como argumento una cadena y devuelve `#t` si los caracteres de la cadena están ordenados de forma ascendente (alfabéticamente). Ejemplos:

```
(ordenada-palabra? "abcdefg") ⇒ #t
(ordenada-palabra? "hola") ⇒ #f
```

Ejercicio 4

Vamos a volver a trabajar con los intervalos de números, continuando con los ejercicios planteados en la primera práctica.

Para **elevar el nivel de abstracción** de nuestras funciones, vamos a **representar un intervalo como una pareja de números**. Así vamos a poder pasar intervalos como parámetros de funciones, devolverlos como resultados de alguna función o guardarlos en listas.

Por ejemplo, el intervalo que empieza en 3 y termina en 12 lo representaremos con la pareja `(3 . 12)`.

a) Define el predicado de la práctica anterior utilizando parejas como parámetros en lugar de números. Define para ello la nueva función `(engloban-intervalos? a b)`. Puedes usar la función `engloba?` definida en la práctica 1 y copiar su definición en esta.

Ejemplos:

```
(define i1 (cons 4 9))
(define i2 (cons 3 10))
(define i3 (cons 12 15))
(define i4 (cons 8 19))

(engloban-intervalos? '(4 . 10) '(5 . 9)) ⇒ #t
(engloban-intervalos? i1 i2) ⇒ #t
(engloban-intervalos? i1 i4) ⇒ #f
```

b) Define las funciones `(union-intervalos a b)` e `(interseccion-intervalos a b)`.

- La función `(union-intervalos a b)` debe devolver el intervalo (pareja) que englobe a los dos.
- La función `(interseccion-intervalos a b)` devolverá la intersección de los intervalos `a` y `b`. En el caso en que no exista intersección, se deberá devolver el símbolo `'vacio`. Puedes usar la función definida en la práctica 1 y copiar su definición en esta.

Ejemplos de `union-intervalos`:

```
(union-intervalos '(4 . 10) '(3 . 8)) ⇒ (3 . 10)
(union-intervalos i2 i3) ⇒ (3 . 15)
```

Ejemplos de `interseccion-intervalos`:

```
(interseccion-intervalos '(4 . 10) '(8 . 15)) ⇒ (8 . 10)
(interseccion-intervalos i1 i2)) ⇒ (4 . 9)
(interseccion-intervalos i1 i3)) ⇒ 'vacio
```

El símbolo `'vacio` también podrá utilizarse como parámetro en las funciones `union-intervalos` y `interseccion-intervalos` para representar un intervalo vacío. Funcionará de la siguiente forma. Supongamos un intervalo cualquiera, por ejemplo `'(8 . 20)`:

```
(union-intervalos 'vacio '(8 . 20)) ⇒ (8 . 20)
(union-intervalos '(8 . 20) 'vacio) ⇒ (8 . 20)
(union-intervalos 'vacio 'vacio) ⇒ 'vacio
(interseccion-intervalos 'vacio '(8 . 20)) ⇒ 'vacio
(interseccion-intervalos '(8 . 20) 'vacio) ⇒ 'vacio
```

Ejercicio 5

Define utilizando la recursión las funciones `(union-lista-intervalos lista-intervalos)` e `(interseccion-lista-intervalos lista-intervalos)` que devuelven el intervalo

(pareja) resultante de la suma o intersección de una lista de intervalos. Debes utilizar las funciones definidas en el ejercicio anterior.

Escribe primero la formulación recursiva del caso general y realiza después la implementación en Scheme.

Ejemplo:

```
(union-lista-intervalos '((2 . 12) (-1 . 10) (8 . 20))) ⇒ (-1 . 20)
(interseccion-lista-intervalos '((12 . 30) (-8 . 20) (13 . 35))) ⇒ (13 . 20)
(interseccion-lista-intervalos '((25 . 30) (-8 . 20) (13 . 35))) ⇒ 'vacio
```
