Práctica 3: Parejas, datos compuestos y listas

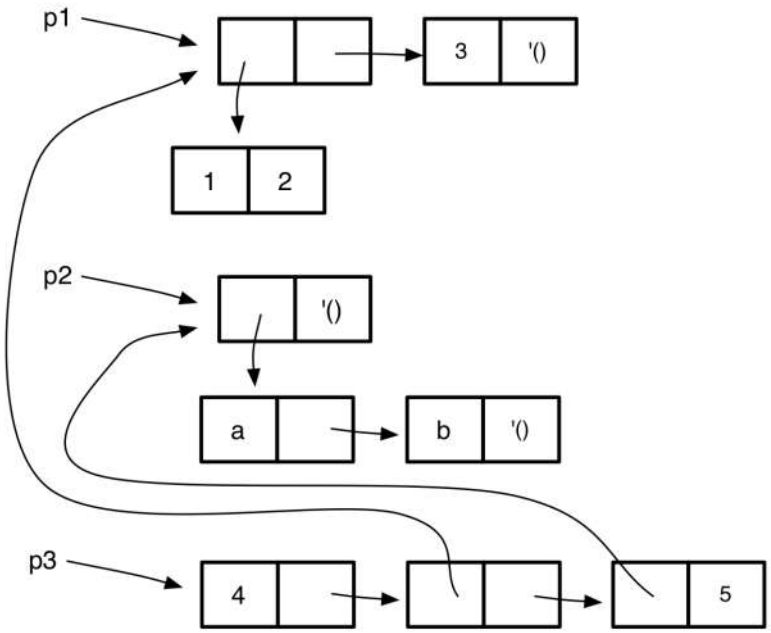
Entrega de la práctica

Para entregar la práctica debes subir a Moodle el fichero `practica03.rkt` con una cabecera inicial con tu nombre y apellidos, y las soluciones de cada ejercicio separadas por comentarios. Cada solución debe incluir:

- La **definición de las funciones** que resuelven el ejercicio.
- Una visualización por pantalla de uno de los ejemplos incluidos en el enunciado que **demuestre qué hace la función**, usando la función de `display`.
- Un conjunto de **pruebas** que comprueben su funcionamiento utilizando la librería `schemeunit`. Estas pruebas deben incluir los ejemplos proporcionados en los ejercicios y un mínimo de **2 casos de prueba sustancialmente distintos** a estos ejemplos.

Ejercicio 1

a) Dado el siguiente *box & pointer*, escribe las sentencias en Scheme (usando llamadas a `list` y `cons`) que definen a `p1`, `p2` y `p3`.



b) Explica si `p1`, `p2` y `p3` son listas y cuántos elementos tienen (en el caso en que lo sean).

c) Escribe las expresiones que:

1. devuelve 3 utilizando `p3`
2. devuelve 5 utilizando `p3`

Ejercicio 2

a) Implementa la función `(intercambia-elem pareja)` que recibe una pareja y devuelve otra pareja con los elementos izquierdo y derecho intercambiados.

Ejemplo:

```
(intercambia-elem (cons 10 5)) ; ⇒ {5 . 10}
```

b) Implementa las funciones `(suma-izq n pareja)` y `(suma-der n pareja)` definidas de la siguiente forma:

- `(suma-izq n pareja)` : recibe un número y una pareja de números, y devuelve una nueva pareja en la que se ha sumado el número n a la parte izquierda de pareja.
- `(suma-der n pareja)` : recibe un número y una pareja de números, y devuelve una nueva pareja en la que se ha sumado el número n a la parte derecha de pareja.

Ejemplos:

```
(suma-izq 5 (cons 10 20)) ; ⇒ {15 . 20}
(suma-der 6 (cons 10 20)) ; ⇒ {10 . 26}
```

c) Implementa la función recursiva `(suma-impares-pares lista-num)` que devuelva una pareja cuya parte izquierda sea la suma de todos los números impares de la lista y la parte derecha la suma de todos sus números pares. En el caso en que ningún número par o impar deberá devolver 0. Debes utilizar las funciones auxiliares definidas en el apartado anterior. También puedes utilizar las funciones predefinidas even? y odd?

Ejemplos:

```
(suma-impares-pares '(3 2 1 4 8 7 6 5)) ; ⇒ {16 . 20}
(suma-impares-pares '(3 1 5)) ; ⇒ {9 . 0}
```

Ejercicio 3

Implementa la función recursiva `(multiplo-de n lista-nums)` que recibe un número `n` y un lista de números y devuelve una lista con los booleanos resultantes de comprobar si cada número de la lista es múltiplo de `n`.

Ejemplo:

```
(multiplo-de 10 '(100 23 10 300 48 7)) ⇒ (#t #f #t #t #f #f)
```

Ejercicio 4

a) Implementa una función recursiva `(filtra-simbolos lista-simbolos lista-num)` que recibe una lista de símbolos y una lista de números enteros (ambas de la misma longitud) y devuelve una lista de parejas. Cada pareja está formada por el símbolo de la i-ésima posición de `lista-simbolos` y el número entero situado esa posición de `lista-num`, siempre y cuando dicho número se corresponda con la longitud de la cadena correspondiente al símbolo. Puedes utilizar las funciones

predefinidas `string-length` y `symbol->string`.

Ejemplo:

```
(filtra-simbolos '(este es un ejercicio de examen) '(2 1 2 9 1 6))
⇒ {{un . 2} {ejercicio . 9} {examen . 6}}
```

b) Escribe la función recursiva `(expande lista-parejas)` que reciba una lista de parejas que contienen un dato y un número y devuelva una lista donde se hayan "expandido" las parejas, añadiendo tantos elementos como el número que indique cada pareja.

Ejemplo:

```
(expande (list (cons #t 3) (cons "LPP" 2) (cons 'b 4)))
⇒ {#t #t #t "LPP" "LPP" b b b b}
```

Ejercicio 5

a) Implementa la función recursiva `(suma-parejas-impar-par lista-parejas)` que recibe una lista de parejas de números enteros, y devuelve una pareja cuya parte izquierda sea la suma de las partes izquierda de las parejas que sean número impar y su parte derecha la suma de las partes derechas que sean número par. Debes utilizar las funciones auxiliares definidas en el apartado 2.b) y también puedes utilizar las funciones predefinidas even? y odd?

Ejemplos:

```
(suma-parejas-impar-par (list (cons 3 2) (cons 6 5) (cons 7 4))) ; ⇒ {10 . 6}
(suma-parejas-impar-par (list (cons 1 5) (cons 4 9) (cons 8 3))) ; ⇒ {1 . 0}
```
