Práctica 4: Funciones como datos de primera clase y funciones de orden superior

Entrega de la práctica

Para entregar la práctica debes subir a Moodle el fichero `practica04.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.
- Un conjunto de **pruebas** que comprueben su funcionamiento utilizando el API `RackUnit`.

Ejercicios

Ejercicio 1

a) Indica qué devuelven las siguientes expresiones, sin utilizar el intérprete. Comprueba después si has acertado.

```
(map (lambda (x)
         (cond
            ((symbol? x) (symbol->string x))
            ((number? x) (number->string x))
            ((boolean? x) (if x "#t" "#f"))
            (else "desconocido"))) '(1 #t hola #f (1 . 2))) ;
⇒ ?

(filter (lambda (x)
            (equal? (string-ref (symbol->string x) 1) #\a))
'(alicante barcelona madrid almería)) ; ⇒ ?

```

```
(foldl (lambda (dato resultado)
         (string-append
          (symbol->string (car dato))
          (symbol->string (cdr dato))
          resultado)) "" '((a . b) (hola . adios) (una . pareja))) ; ⇒ ?

(foldr (lambda (dato resultado)
           (cons (+ (car resultado) dato)
                 (+ (cdr resultado) 1))) '(0 . 0) '(1 1 2 2 3 3)) ; ⇒ ?
```

b) Sin utilizar el intérprete DrRacket, rellena los siguientes huecos para obtener el resultado esperado. Después usa el intérprete para comprobar si has acertado.

```
; Los siguientes ejercicios utilizan esta definición de lista

(define lista '((2 . 7) (3 . 5) (10 . 4) (5 . 5)))


; Queremos obtener una lista donde cada número es la suma de las
; parejas que son pares

(filter ________
        (________ (lambda (x) (+ (car x)
                                 (cdr x)))
                 lista))
; ⇒ (8 14 10)

; Queremos obtener una lista de parejas invertidas donde la "nueva"
; parte izquierda es mayor que la derecha.

(filter ___________
        (map ____________ lista))
; ⇒ ((7 . 2) (5 . 3))

; Queremos obtener una lista cuyos elementos son las partes izquierda
; de aquellas parejas cuya suma sea par.

(foldr __________ '()
        (_________ (lambda (x) (even? (+ (car x) (cdr x))))
lista))
; ⇒ (3 10 5)
```

c) Rellena los siguientes huecos **con una única expresión**. Comprueba con el intérprete si lo has hecho correctamente.

```
(define (f x) (lambda (y z) (string-append y z x)))
(define g (f "a"))
(check-equal? ____________________ "claselppa")



(define (f x) (lambda (y z) (list y x z)))
_____________
(check-equal? (g "hola" "clase") (list "hola" "lpp" "clase"))


(define (f g) (lambda(z x) (g z x)))
(check-equal? _____________________  '(3 . 4))
```

Ejercicio 2

Implementa utilizando funciones de orden superior las funciones `(crea-baraja lista-parejas)` y `(expande-lista lista-parejas)` de la práctica 3. Para la implementación de `expande-lista` debes utilizar la función `expande-pareja` usada también en la práctica 3.

```
(crea-baraja '((#\u2660 . #\A) (#\u2663 . #\2)
               (#\u2665 . #\3) (#\u2666 . #\R)))
; ⇒ (A♠ 2♣ 3♥ R♦)

(expande-lista '((#t . 3) ("LPP" . 2) (b . 4)))
; ⇒ '(#t #t #t "LPP" "LPP" b b b b))
```

Ejercicio 3

a) Implementa usando funciones de orden superior la función `(suma-n-izq n lista-parejas)` que recibe una lista de parejas y devuelve otra lista a la que hemos sumado `n` a todas las partes izquierdas.

Ejemplo

```
(suma-n-izq 10 '((1 . 3) (0 . 9) (5 . 8) (4 . 1)))
; ⇒ ((11 . 3) (10 . 9) (15 . 8) (14 . 1))
```

b) Implementa usando funciones de orden superior la función `(aplica-2 func lista-parejas)` que recibe una función de dos argumentos y una lista de parejas y devuelve una lista con el resultado de aplicar esa función a los elementos izquierdo y derecho de cada pareja.

Ejemplo:

```
(aplica-2 + '((2 . 3) (1 . -1) (5 . 4)))
; ⇒ (5 0 9)
(aplica-2 (lambda (x y)
            (if (even? x)
                y
                (* y -1))) '((2 . 3) (1 . 3) (5 . 4) (8 . 10)))
; ⇒ (3 -3 -4 10)
```

c) Implementa la función `(filtra-simbolos lista-simbolos lista-num)` de de la práctica 3, usando una composición de funciones en las que se use `map`.

Ejercicio 4

a) La función de Racket `(index-of lista dato)` devuelve la posición de un dato en una lista o `#f` si el dato no está en la lista. Si el dato está repetido en la lista devuelve la posición de su primera aparición.

Implementa la función `mi-index-of` que haga lo mismo, usando funciones de orden superior. Puedes usar también alguna función auxiliar.

Pista

Puedes utilizar la función `foldl` para recorrer la lista de izquierda a derecha buscando el dato. Puedes usar como resultado del `foldl` una pareja en cuya parte derecha vayamos calculando la posición y en la parte izquierda haya un booleano que indique si hemos encontrado o no el dato.

Ejemplos:

```
(mi-index-of '(a b c d c) 'c) ; ⇒ 2
(mi-index-of '(1 2 3 4 5) 10) ; ⇒ #f
```

b) Completa la definición de la siguiente función de orden superior `(busca-mayor mayor? lista)` que busca el mayor elemento de una lista. Recibe un predicado `mayor?` que compara dos elementos de la lista y devuelve `#t` o `#f` dependiendo de si el primero es mayor que el segundo.

```
(define (busca-mayor mayor? lista)
  (foldl __________ (car lista) (cdr lista)))
```

Escribe algunos `check-equal?` en los que compruebes el funcionamiento de `busca-mayor`, utilizando funciones `mayor?` distintas.

c) Define la función `(posicion-mayor mayor? lista)` que devuelva la posición del mayor elemento de la lista utilizando las dos funciones anteriores.

Ejercicio 5

a) Supongamos que vamos a representar una mano de cartas como una lista de cartas. Podemos entonces representar un juego de *n* manos como una lista de *n* listas.

Por ejemplo, la siguiente lista representaría un juego con 3 manos de 5 cartas:

```
((K♦ J♥ 2♥ 2♠ 8♥) (A♥ 3♠ 5♦ 3♣ J♦) (3♦ Q♥ 0♠ 2♣ 9♦))
```

Para construir este juego vamos a necesitar una función auxiliar que vaya construyendo las manos, añadiendo una carta a la primera mano de la lista si ésta no tiene todas las cartas necesarias.

Tienes que definir la función `(añade-carta carta n-cartas manos)` que recibe una carta, un número que representa el número de cartas que deben tener las manos y una lista de manos, cuya primera mano puede estar incompleta. En ese caso, la función añadirá la carta a esa primera mano de la lista. Si la primera mano está completa, o la lista de manos está vacía, se deberá añadir una nueva mano a la lista, con la única carta que se pasa como parámetro.

Ejemplos:

```
(añade-carta 'K♦ 3 '()) ; ⇒  ((K♦))
(añade-carta 'J♥ 3 '((2♥ 2♠) (5♦ 3♣ J♦))) ; ⇒ ((J♥ 2♥ 2♠) (5♦
```

```
3♣ J♦))
(añade-carta '3♠ 3 '((5♦ 3♣ J♦))) ; ⇒ ((3♠) (5♦ 3♣ J♦))
```

Una vez definida la función anterior, y utilizando funciones de orden superior, debes implementar la función `(reparte n-manos n-cartas baraja)` que devuelva un juego con *n* manos de *n* cartas sacadas la parte de arriba de una baraja inicial. Puedes utilizar las funciones `baraja-poker` y `mezcla` de la práctica anterior para crear la baraja inicial.

Ejemplo:

```
(define baraja (mezcla (baraja-poker)))
baraja ; ⇒ (Q♠ 9♥ 4♣ K♠ 7♥ J♦ 5♠ 4♠ 5♦ 6♦ 8♦ A♦ Q♦ 0♣ 7♦ 3♠ Q♥
...)
(reparte 3 5 baraja)
; ⇒ ((7♦ 0♣ Q♦ A♦ 8♦) (6♦ 5♦ 4♠ 5♠ J♦) (7♥ K♠ 4♣ 9♥ Q♠))
```

b) La siguiente función devuelve el valor de una carta:

```
(define (valor-carta carta orden)
  (+ 1 (index-of orden (string-ref (symbol->string carta) 0))))
```

El parámetro `orden` es una lista de todos los caracteres que representan los posibles valores de una carta, ordenados de menor a mayor.

Por ejemplo:

```
(define orden '(#\2 #\3 #\4 #\5 #\6 #\7 #\8 #\9 #\0 #\J #\Q #\K
#\A))
(valor-carta 'A♠ orden) ; ⇒ 13
(valor-carta 'J♥ orden) ; ⇒ 10
(valor-carta '2♦ orden) ; ⇒ 1
```

Implementa, utilizando funciones de orden superior y funciones definidas anteriormente en esta práctica, la función `(mano-ganadora lista-manos)` que recibe una lista de manos y devuelve la posición de la mano ganadora utilizando la valoración del póker. La mano ganadora es la que tiene una carta más alta. Si hay empate, deberás devolver la posición de la primera mano que participa en el empate.

Pista

Puedes definir una función de un único parámetro que devuelve el valor de una carta usando el orden definido por el póker usando la siguiente expresión lambda:

```
(lambda (carta)
     (valor-carta carta
          '(#\2 #\3 #\4 #\5 #\6 #\7 #\8 #\9 #\0 #\J #\Q #\K #\A)))
```

Puedes usar cualquier función definida anteriormente.

```
(define lista-manos (reparte 3 5 (mezcla (baraja-poker))))
; lista-manos ⇒ ((9♦ 2♦ K♥ 0♦ 7♥) (6♦ 4♠ 7♣ 5♥ 4♦) (0♣ 4♣ 5♠
3♥ J♥))
(mano-ganadora lista-manos) ; ⇒ 0
```

Domingo Gallardo, Cristina Pomares, Antonio Botía, Francisco Martínez