v.7.6

Práctica 4 - Estructuras

Programación I

1 Coordenadas: la estructura `posn`

En la clase de teoría vimos que podíamos representar estructuras con multiples campos. En particular vimos como representar coordenadas en el plano a través de una estructura con dos campos. En *racket* viene definida la estructura `posn`, que podemos usar a tal fin. Aquí el diseño de datos de la misma:

```
(define-struct posn [x y])
 ; posn es (Number , Number)
 ; interpretación: un elemento en posn representa una
 ; posición en coordenadas cartesianas
```

Recordemos también que esta definión crea automáticamente cuatro funciones para operar con la estructura `posn`:

- `make-posn`, llamado **constructor**. Es el que nos permite crear objetos de tipo `posn`. Su signatura es

  ```
  ;  make-posn : Number Number -> posn
  ```

- `posn-x` y `posn-y`, llamados **selectores**. Nos permiten observar cada uno de los campos de la estructura. En este caso ambas funciones tienen la misma signatura:

  ```
  ;  posn-x : posn -> Number
  ;  posn-y : posn -> Number
  ```

- `posn?`, llamado el **predicado** de la estructura. Dado un objeto cualquiera, devuelve `#true` si su argumento es un objeto de tipo `posn`.

A través de dos leyes fundamentales, podemos calcular expresiones que contengan estructuras. Estas leyes son simples y relacionan cómo se comportan los selectores respecto del constructor:

> Las (ley 1) y (ley 2) nos dice que al evaluar posn-x y posn-y obtenemos la primer "a" y la segunda "b" componente del objero creado por make-posn respectivamente

```
(posn-x (make-posn a b))
== definición de posn-x (ley 1)
a
```

```
(posn-y (make-posn a b))
== definición de posn-y (ley 2)
b
```

A partir de estas dos leyes, podemos calcular expresiones con estructuras de tipo `posn`. Por ejemplo, si tenemos:

```
(define p (make-posn 3 4))
(define q (make-posn -2 0.5))
```

Podemos calcular la expresión `(posn-y (make-posn (posn-y q) (posn-x p)))`:

```
(posn-y (make-posn (posn-y q) (posn-x p)))

== definición de q
```

```
(posn-y (make-posn (posn-y (make-posn -2 0.5)) (posn-x p)))

== (ley 2)

(posn-y (make-posn 0.5 (posn-x p)))

== definición de p

(posn-y (make-posn 0.5 (posn-x (make-posn 3 4))))

== (ley 1)

(posn-y (make-posn 0.5 3))

== (ley 2)

3
```

Ejercicio 1. Calcule el valor de las siguientes 3 expresiones:

- `(posn-x p)`
- `(- (posn-y p) (posn-y q))`
- `(posn-y (make-posn (posn-x p) (posn-x q)))`

> Recuerde que al decir **calcule** le estamos pidiendo resolver *con papel y lápiz*, y no utilizando *racket* .

Luego revise sus procedimientos (en especial el último) con el evaluador paso a paso de *racket* .

Ejercicio 2. Defina una función `dist-origen` que calcule la distancia al origen de un punto, representado por una estructura `posn`. Calcule el valor de las siguientes expresiones:

- `(dist-origen (make-posn (/ 6 2) 4))`
- `(+ (dist-origen (make-posn 12 5)) 4)`

Ejercicio 3. Diseñe una función `simétrico`, que dada una posición, nos devuelva su simétrica respecto del origen.

Ejercicio 4. Diseñe una función `distancia`, que dado dos puntos en el plano calcule la distancia entre ellos. Extienda la defínción para que, en caso que alguno de los argumentos no sea de tipo `posn`, muestre el mensaje `"Tipos incorrectos para la función."`

Ejercicio 5. Usando una estructura `posn` como estado, modifique el ejercicio 4 de la Práctica 3 para que el objeto pueda moverse tanto vertical como horizontalmente, respondiendo a las teclas de dirección habituales. El comportamiento respecto al mouse debe ser el siguiente: Cuando se presiona un botón, el objeto debe aparecer en la coordenada donde se produjo el evento.

Ejercicio 6. En este ejercicio moveremos una mosca a través de la escena. El estado del sistema será una estructura posn, que representa la posición sobre la que se va a dibujar la mosca. Defina constantes para representar el ancho y alto de la escena, así como también aquellas que permiten dibujar la mosca. Genere una expresión big-bang tal que:

- El estado inicial del sistema es el centro de la imagen.
- La función que responde a la cláusula `to-draw` dibuja una mosca en la posición determinada por el estado actual.
- Agregue la condición de que por cada tick del reloj la mosca se mueve aleatoriamente una distancia `DELTA`, tanto verticalmente como horizontalmente. Es decir, para cada dirección, la mosca puede desplazarse `DELTA` o `-DELTA` unidades, dando lugar a cuatro resultados posibles para el nuevo estado.

  La función random, dado un entero positivo k, nos retornará un número entero pseudo-aleatorio en el rango: [0, k). Por ejemplo:

  ⁞

```
> (random 5)
3
> (random 5)
2
```

Observemos que si evaluamos la función `(random 2)` , podemos utilizar el resultado binario para tomar decisiones aleatorias, por ejemplo, podemos crearnos la siguiente función:

```
> (define (elegir-random a b) (if (= (random 2) 1) a b))
```

Que nos permite elegir aleatoriamente entre dos valores.

```
> (elegir-random "big" "bang")
"big"
```

Utilice dicha función, para implementar la condición sobre el tick del reloj.

- Modifique la función anterior para asegurarse de que la mosca no sale fuera de escena. A este punto, ejecutando el programa, deberíamos ver una mosca que se mueve aleatoriamente a través de la escena.
- En cada click del mouse, compare la posición del evento, con la de la mosca. Si se encuentran a una distancia menor a un `GAMMA`, se pasará a un estado final, distinto al resto. Por ejemplo, puede ser la posición (-1,-1), pues nunca se alcanza dicho estado en el funcionamiento del programa.

  ```
  (define ESTADO-FINAL (make-posn -1 -1))
  ```

- Modifique la función asociada a la cláusula `to-draw`, de manera que al alcanzar el estado final, imprima el texto `"MOSCA ATRAPADA"` en pantalla.
- Configure la cláusula `stop-when`, para que al alcanzar dicho estado final, termine el programa.

Observación: Si le parece que la mosca se mueve demasiado rápido, busque en la documentación de `DrRacket` lo referente a `rate-expr` en la cláusula `on-tick`.

2 Más estados

Ejercicio 7. En el ejercicio 7 de la práctica 3, la velocidad del auto era siempre la misma: 3 píxeles por cada tick del reloj. Considere ahora la siguiente definición para representar el estado del sistema:

```
(define-struct Auto [hpos vel])
; Auto es (Number, Number)
; Intepretación: El primer elemento es la posición horizontal del auto,
mientras que el segundo representa la velocidad expresada en píxeles sobre tick.
```

Modifique el programa para que el estado del sistema sea representado por esta nueva estructura. El manejador del teclado debe adaptarse de acuerdo a la siguiente especificación:

- La tecla `"up"` incrementa la velocidad del auto, mientras que la tecla `"down"` la decrementa; de acuerdo a una constante `DELTA-VEL`

Ejercicio 8. Considere la siguiente definición de una estructura para nuestro editor de texto:

```
(define-struct Texto [s color tam])
; Texto es (String, Color, Number)
; Intepretación: El primer elemento es la cadena a mostarse, mientras que el segundo y
el tercero determinan el color y tamaño de la fuente en píxeles respectivamente.
```

Modifique el ejercicio del editor de textos de acuerdo a lo siguiente:

- La cadena se muestra con el color y tamaño que determina el estado actual. Decida usted qué color y tamaño serán el inicial.

- Como en la práctica anterior, cada vez que se presiona una letra o número este debe agregarse a la cadena que se está mostrando. El comportamiento para la tecla `"backspace"` debe mantenerse.
- Elija 6 colores de su preferencia, y modifique su programa para que al presionar las teclas `"f1"`, `"f2"`, ..., `"f6"`, el color del texto cambie a estos colores.
- Al presionar la tecla `"up"`, la fuente debe incrementarse, mientras que al presionar la tecla `"down"`, la fuente debe achicarse (siempre que sea posible).

---

3 Estructuras diversas

Ejercicio `9`. En este ejercicio representaremos un auto con una estructura con los siguientes campos:

- 1er campo: Modelo.
- 2do campo: Año.
- 3to campo: Tipo de combustible (diesel o nafta).
- 4to campo: Rendimiento óptimo, expresado en kilómetros por litro.

Teniendo en cuenta esto se pide que:

- Diseñe una estructura `Auto` que contenga los campos descriptos más arriba.
- Diseñe una función `costo-viaje` que tome como entrada un valor de tipo `Auto` y un número de kilómetros a recorrer y calcule el costo del viaje. Para esto debe tener en cuenta:
  - Cantidad de combustible: el número de litros necesarios para recorrer `m` kilómetros con un auto nuevo está determinado por su rendimiento óptimo. Sin embargo, con el correr de los años, el rendimiento disminuye. Se estima que si el auto tiene:
    - Entre 1 y 5 años, el rendimiento disminuye 2%.
    - Entre 6 y 10 años, el rendimiento disminuye 6%.
    - Entre 10 y 15 años, el rendimiento disminuye 10%.
    - Más de 15 años, el rendimiento disminuye 15%.

    Es decir, si un auto 0km rinde 13km/litro, después de un año rendirá 12.74 km/litro, y después de 12 años 11,7 km/litro.
  - Peajes: por cada 100 kilómetros recorridos, se debe pagar un peaje de $50.
  - Precio combustible: el precio actual del litro de nafta es $19 y el litro de diesel $17.

  Ejemplo: Un gol naftero 2013 de rendimiento óptimo 13km/litro, debido a sus 4 años de antigüedad tendrá un rendimiento de 12,74 km/litros. Por lo tanto, recorrer 450 kms tendrá un costo de: (450 / 12,74) * $19 + $200

Ejercicio `10`. En este ejercicio representaremos un `Alumno` con una estructura con los siguientes campos:

- 1er campo: Nombre del alumno.
- 2do campo: Promedio de sus calificaciones (un valor entre 0 y 10).
- 3er campo: Porcentaje de asistencia a clases (un valor entre 0 y 100).

Teniendo en cuenta esto se pide:

- Diseñe una estructura `Alumno` que contenga los campos descriptos más arriba.

- Diseñe una función `condicion` que tome como entrada un valor de tipo `Alumno` y devuelva un string indicando su condición. Condiciones posibles: `"Libre"`, `"Regular"` y `"Promovido"`.

  Para calcular la condición del alumno deben tenerse en cuenta las siguientes reglas:

  - Si el alumno tiene un porcentaje de inasistencia mayor al 40% queda automáticamente libre, sin importar el promedio de sus calificaciones.
  - Si el alumno tiene una asistencia mayor o igual al 60%:
    - y tiene una nota inferior a 6, también se considera libre.
    - y tiene una nota mayor o igual a 6 y menor estricta que 8, se considera regular.
    - y una nota mayor o igual a 8, se considera promovido.

  En caso en que la función `condicion` reciba como entrada un dato que no corresponda a una estructura `Alumno` deberá responder con un mensaje de error (como por ejemplo: `"Tipo de dato inválido"`).

Ayuda: Recuerde que cada estructura que define viene acompañada de un predicado que determina si un objeto es o no una estructura de ese tipo.

Ejercicio `11`. Representaremos una casa con una estructura con los siguientes campos:

- 1er campo: *propietario*.
- 2do campo: *dirección*.
- 3er campo: *superficie en metros cuadrados*.
- 4to campo: *zona*.

Teniendo en cuenta esto se pide:

- Diseñe una estructura `Casa` que contenga los campos descriptos más arriba.
- Diseñe una función `venta` que tome como entrada un valor de tipo `Casa` y devuelva un mensaje sobre los datos de la venta de dicha propiedad. Dicho mensaje deberá dar los datos sobre el propietario que vende, en qué dirección se encuentra y el monto de dinero que recibe el propietario por dicha venta, luego de realizados los descuentos correspondientes por el sellado de la escritura. Para realizar los cálculos se deben tener en cuenta las siguientes cuestiones:
  - Existen 4 zonas:
    - Zona A: el metro cuadrado en esta zona tiene un valor de 20000 pesos
    - Zona B: el metro cuadrado en esta zona tiene un valor de 15000 pesos
    - Zona C: el metro cuadrado en esta zona tiene un valor de 10000 pesos
    - Zona D: el metro cuadrado en esta zona tiene un valor de 5000 pesos
  - El propietario debe pagar un sellado que se descontará del precio de la venta. Si el precio de la venta supera el millón de pesos el sellado tiene un costo del 5 % del valor y si no lo supera se debe abonar el 3% del valor de la propiedad.

Ejemplo:

- Supogamos que el señor José Romero vende una propiedad que se encuentra en la calle Rueda 3456. Dicha propiedad posee una superficie de 120 metros cuadrados y, por la dirección en que se encuentra, pertenece a la zona C. El monto de dinero que José recibe por dicha venta es de 1140000 pesos. Si aplicáramos la función `venta` en este caso desearíamos que esta función nos

devolviera el siguiente mensaje: `"El señor José Romero recibirá 1140000 pesos por la venta de su propiedad ubicada en la calle Rueda 3456."`

En caso que la función `venta` reciba como entrada un dato que no sea de tipo `Casa` deberá mostrar el mensaje `"Tipo de dato incorrecto"`.

En caso que la función `venta` reciba como entrada un dato de tipo `Casa` con una zona distinta a A, B, C o D deberá mostrar el mensaje `"No se puede calcular el precio de venta por no disponer de los valores del metro cuadrado para la zona solicitada"`.

Ejercicio 12. En este ejercicio representaremos una persona mediante una estructura con 5 campos:

- 1er campo: el nombre y apellido.
- 2do campo: el valor numérico de su peso.
- 3er campo: un string que representa la unidad en la cual está dado el peso (valores posibles: "g" o "kg").
- 4to campo: el valor numérico de la estatura.
- 5to campo: un string que representa la unidad en la cual está dada la estatura (valores posibles: "m" o "cm").

Teniendo en cuenta esto se pide:

- Diseñe una estructura `Persona` que le permita representar a cualquier persona con su peso y estatura.
- Diseñe una función `IMC` que tome como entrada un valor de tipo `Persona` y calcule su índice de masa corporal. En caso que no reciba como entrada una estructura de tipo `Persona` deberá mostrar el siguiente mensaje de error: `"Tipo de dato inválido"`.

Ayuda: Por si no lo sabe, el índice de masa corporal (IMC) de una persona se calcula según la siguiente fórmula:

$$IMC = \frac{\text{Peso (kg)}}{\text{Estatura}^2\text{(Mts.)}}$$

Ayuda: Le puede servir conocer las siguientes equivalencias:

1 kg = 1000 g

1 m = 100 cm