Curso: Lenguajes de programación
Practica 2: Tipo de Datos Abstractos

Karla Ramírez Pulido Alan Alexis Martínez Lopez José Ricardo Desales Santos

José Eliseo Ortíz Montaño

Fecha de inicio: Martes 07 de Marzo del 2023.
Fecha de entrega: Martes 21 de Marzo del 2023.

Objetivo

Familiarizarse con la definición y uso de *tipos de datos abstractos* (TDA)[1] en Racket, así como practicar el uso de la recursión y profundizar en el uso de las funciones nativas del lenguaje.

Estructura

La práctica está conformada por el siguiente par de archivos:

- `datatypes.rkt`, en el cual se encuentran definidos los TDA `pilas-y-colas` y `binary-search-tree`. En este mismo archivo, tienes que definir el TDA `List`, el cual corresponde al ejercicio 4.1 de esta práctica.
- `practica2.rkt`; aquí es donde deberás realizar el resto de implementaciones indicadas en cada uno de los ejercicios.

Es importante que ambos archivos los guardes dentro de un mismo directorio, ya que de lo contrario tendrás problemas al ejecutar el interprete debido a que el archivo `practica2.rkt` importa al archivo `datatypes.rkt`

Ejercicios

1. Investiga las siguientes funciones nativas del lenguaje. Deberás especificar sus firmas y dar 2 ejemplos de uso de cada una de ellas:
    1. `filter`
    2. `foldr`
    3. `foldl`
2. En clase, definimos el TDA *pilas-y-colas* y los respectivos constructores necesarios.

```
;;Definición del tipo Nodo
(define-type Nodo
    [Void]
    [nodo (elem number?) (siguiente Nodo?)])

;;Defínción del tipo pilas-y-colas
(define-type pilas-y-colas
    [Pila (nodos Nodo?)]
    [Cola (nodos Nodo?)])
```

[1] Abstract Data Type, ADT.

1. Define la el predicado `(contains? struct elem)` que recibe una estructura de datos y un elemento cualquiera, determina si dicho elemento se encuentra almacenado en la estructura pasada como parámetro.
2. Define la función `(mira struct)` la cual recibe una estructura (`struct`) y regresa el elemento que se encuentra en el tope de la estructura, i.e. el elemento que sería eliminado de la estructura al llamar a la función `(saca struct)`.
3. Define la función `(size struct)` la cual determina la cantidad de elementos en la estructura. Si la estructura es vacía, el número de elemento es igual a cero.

3. En clase, definimos el tipo de dato abstracto árbol binario de búsqueda *BST* (*Binary Search Tree*) de la siguiente manera:

```
;;Definición del tipo BST
(define-type binary-search-tree
[VoidTree]
[BST (elem number?) (lt binary-search-tree?) (rt binary-search-tree?)])
```

   donde todos los elementos del subárbol derecho son menores o iguales a la raíz del árbol y todos los elementos del subárbol izquierdo son estrictamente mayores a esta.

   1. Define la función `(delete-bst bst elem)` que dado un BST y un elemento, lo elimina del BST y entrega como resultado el BST producto de realizar dicha operación.
   2. Define la función `(count-leaves-bst bst)` que dado un BST, devuelve el número de hojas de éste. Recuerda que una hoja es aquel nodo que tanto su subárbol izquierdo como derecho son void[2].
   3. Define la función `(map-bst fun bst)` tal que da como resultado el BST que tiene como elementos al los elementos del árbol pasado como parámetro después de haberles aplicado `fun` a cada uno de ellos.

4. Define el tipo de dato abstracto `List`, el cual representa nuestra implementación de listas.
   Algunos ejemplos son:

```
(Empty)
(Cons 345 (Empty))
(Cons 1 (Cons 2 (Cons 3 (Cons 4 (Cons 5 (Empty))))))
```

   1. Define el tipo de dato abstracto acorde a la especificación anterior. Recuerda que esta definición se tiene que realizar en el archivo `datatypes.rkt`, ya que es ahí donde tenemos el resto de definiciones de los TDA que hemos trabajado.
   2. Define las siguientes funciones/predicados:
      a) `(contains-List? Lst elem)`
      b) `(length-List Lst)`
      c) `(add-List elem Lst)`
      d) `(delete-List Lst)`
      e) `(map-List fun Lst)`
      f) `(filter-List pred Lst)`

      la especificación de cada una de estas funciones corresponde a la de sus símiles para listas nativas del lenguaje.

[2] De acuerdo con nuestra definición, sería de tipo `VoidTree`.

Extra

1. **(+ 1.2 pts)** Define la función `(cool-print struct)` tal que dado un ejemplar de `pilas-y-colas` imprima en pantalla una representación gráfica "más amigable" de la estructura pasada como parámetro.[3]
   Ejemplo:

```
;;Definiendo algunas pilas y colas útiles para el ejemplo
(define p
        (Pila (nodo 1 (nodo 2 (nodo 3 (nodo 4 (nodo 5 (Void))))))))

((define c
        (Cola (nodo 323 (nodo 2342 (nodo 32112 (Void))))))

;;Llamando a la cool-print en el área de interacciones
>(cool-print p)
[1 2 3 4 5] <->

>(cool-print c)
<-[323 2342 32112]<-
```

[3] Observa que no se está regresando como resultado una cadena.

Entrega

Acerca de la entrega de la práctica:

- La realización y entrega de la práctica deberá realizarse en equipos de a lo más 4 personas.[4]
- Su entrega debe consistir en los archivos `datatypes.rkt` y `practica2.rkt`, los cuales se proporcionan junto con este archivo. Recuerden agregar, en cada uno de estos, los datos de los integrantes del equipo.
- La entrega debe respetar el orden de las funciones que es especificado en este documento. El uso de funciones auxiliares está totalmente permitido, sin embargo, deben ser declaradas justo después de la función principal que hace uno de ellas.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Ejercicio 1: [Descripción del ejercicio]
(define (my-fun a b c d) ... )

;; [Documentación de la función auxiliar - incluír una breve descripción de
;; la motivación de la función]
(define (an-aux-fun lst1 lst2) ... )
...
...
...
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

- Las funciones deben incluir comentarios, tanto de documentación como del proceso de desarrollo. Estos deben ser claros, concisos y descriptivos.
- Queda estrictamente prohibido utilizar funciones primitivas del lenguaje que resuelvan directamente los ejercicios.[5]
- Se deberá subir la versión final de su práctica al apartado del classroom correspondiente antes de la fecha límite. Esto sólo debe realizarlo un integrante del equipo; el resto deberá marcar como entregada la actividad y en un comentario privado especificar quienes son los miembros del equipo.
- Para la administración de la práctica en GitHub, selecciona el siguiente enlace.

[4] Cualquier situación con respecto a este punto será tratada de acuerdo a las particularidades del caso. Para esto, acercarse al ayudante del rubro del laboratorio a la brevedad.

[5] Cualquer duda con respecto a este punto, por favor manda un correo para atenderla.