Trabajo Práctico 3. Programación Funcional.

Puntuación

Puntaje Total: 100 puntos.
Aprobación: 60 puntos.
Fecha de entrega: 16/12/2023.

Condiciones de entrega

1. El presente trabajo práctico deberá resolverse en grupo de hasta 3 (tres) personas.
2. Entrega: Se realizará por medio del Campus Virtual de la UTN, en la tarea correspondiente al TP 3. La extensión del archivo será .zip o .rar, de acuerdo al programa de compresión usado. El nombre del archivo se consigue concatenando un prefijo del número del TP con los apellidos de los integrantes separados por guiones (Ej: Pérez y Abdala, el nombre será `tp3-abdala-perez.zip`). Note que no hay espacios en blanco ni acentos en el nombre de archivo. Dentro del archivo de entrega, deben constar los siguientes:
    - Fuentes DrRacket: Se debe entregar un archivo denominado `tp3.rkt`
    - Los casos de prueba se entregarán en un archivo de texto, no deben ser capturas de pantalla. Deberán cubrir diferentes resultados que puedan obtenerse de la evaluación de las funciones solicitadas. Se enfatiza que se adjunten casos de prueba que sean claros, válidos y suficientes para poder probar el trabajo. Entregar este archivo con el nombre `casos-de-prueba.txt`.
    - Archivo de texto (`INTEGRANTES.txt`) con una línea para cada integrante en la cual figure el nombre del alumno/a y su dirección de email.
4. Penalización por entrega fuera de término: Si el trabajo práctico se entrega después de la fecha indicada, y hasta una semana tarde, tendrá una quita de 15 puntos. No serán recibidos trabajos luego de una semana de la fecha de entrega. Los trabajos que se deban rehacer/corregir fuera de la fecha de entrega tienen una quita de 30 puntos.

Enunciado

Un array asociativo (AA) es un tipo de dato abstracto que permite almacenar una colección de pares (clave, valor). Las implementaciones de este tipo de datos varía con los lenguajes de programación. Para el trabajo práctico se considera que:

- Una clave puede estar a lo sumo en un par de un AA.
- Un valor puede estar en más de un par de un AA.
- Las claves en un AA pueden ser de distinto tipo.
- Los valores de un AA pueden ser de distinto tipo.

- Los pares en un AA no tienen un orden especificado.

A continuación se presentan ejemplos de arrays asociativos. Para clave y valor se emplean tipos de datos de Scheme, el resto de la sintaxis que sigue es solo ilustrativa:

```
array-1 = { (8.6, '(racket rkt)),
            (5.07, '(prolog pl)) }

array-2 = { (#\a, 97),
            (#t, 'boolean),
            (#f, 'boolean),
            (+, "suma"),
            ('(), "lista vacia") }

array-3 = { ("impar", odd?),
            ("ultimo", (lambda (lista) (car (reverse lista))))}
```

En el **ejemplo de implementación** disponible en la última sección de este documento se muestra la construcción de estos arrays asociativos en Scheme.

El **objetivo** del trabajo práctico es implementar funciones en Scheme para realizar operaciones de inserción, búsqueda y eliminación sobre arrays asociativos.

1. Representación

Un array asociativo es representado como una lista, donde el primer elemento es la etiqueta array-asociativo y el resto de los elementos son pares punteados (“dotted pairs”) de Scheme. En cada par, el primer elemento corresponde a la clave y el segundo elemento corresponde al valor.

Para la creación de un array asociativo se define la función `array-asoc-crea`:

```
(define array-asoc-etiqueta 'array-asociativo)
(define (array-asoc-crea) (list array-asoc-etiqueta))
```

Además, se define el símbolo `nil` que será empleado como retorno de algunas de las funciones solicitadas:

```
(define nil 'nil)
```

Las definiciones anteriores `array-asoc-etiqueta`, `array-asoc-crea` y `nil` deben ser incluidas en el programa junto a las funciones solicitadas.

2. Funciones solicitadas

Se solicita que defina las siguientes funciones en Scheme R5RS. Puede agregar las funciones auxiliares que considere necesarias, como así también usar las funciones de la Plantilla de Scheme disponible en el Campus Virtual.

1. **`(elimina array clave)`**: devuelve un nuevo array asociativo con los mismos pares que `array`, excepto aquel par con `clave`. Si la `clave` no existe en `array`, el array asociativo retornado tiene los mismos elementos que `array`.

Ejemplos:

```
> (elimina array-3 "ultimo")
(array-asociativo ("impar" . #<procedure:odd?>))
> (elimina array-1 11)
(array-asociativo (8.6 racket rkt)[1] (5.07 prolog pl))
```

2. **`(agrega-reemplaza array clave valor)`**: retorna un nuevo array asociativo resultante de agregar el par `(clave . valor)` en `array`. En caso de que la `clave` existiese en `array`, el valor del par es reemplazado por `valor`.

Ejemplos:

```
> (agrega-reemplaza (array-asoc-crea) '(1) 2)
(array-asociativo ((1) . 2))
> (agrega-reemplaza array-1 11 '(pharo st))
(array-asociativo (8.6 racket rkt) (5.07 prolog pl) (11 pharo st))
> (agrega-reemplaza array-3 "impar" nil)
(array-asociativo ("ultimo" . #<procedure:...>) ("impar" . nil))
```

3. **`(claves array)`**: devuelve una lista con las claves del array asociativo `array`.

Ejemplos:

```
> (claves array-1)
(8.6 5.07)
> (claves array-2)
(#\a #t #f #<procedure:+> ())
```

4. **`(valores array)`**: retorna una lista con los valores del array asociativo `array`.

Ejemplos:

```
> (valores array-1)
((racket rkt) (prolog pl))
> (valores array-3)
(#<procedure:odd?> #<procedure:...>)
```

[1] Representa el par punteado `(8.6 . (racket rkt))`.

5. **`(incluye-clave? array clave)`**: retorna `#t` si el `array` incluye la `clave`; e. o. c.[2] `#f`.

Ejemplos:

```
> (incluye-clave? array-2 +)
#t
> (incluye-clave? array-3 #\b)
#f
```

6. **`(incluye-valor? array valor)`**: retorna `#t` si `array` incluye al menos un par con segundo elemento `valor`; e. o. c. `#f` .

Ejemplos:

```
> (incluye-valor? array-3 1)
#f
> (incluye-valor? array-2 'boolean)
#t
```

7. **`(valor-en array clave)`**: devuelve el segundo elemento (valor) del par de `array` cuyo primer elemento es `clave`; e. o. c. `nil`.

Ejemplos:

```
> (valor-en array-2 #t)
boolean
> (valor-en array-3 "impar")
#<procedure:odd?>
> (valor-en array-3 1)
nil
```

8. **`(selecciona array predicado)`**: dada la función de un argumento `predicado`, retorna una lista con los valores de `array` cuya evaluación como único argumento de `predicado` devolvió `#t`.

Ejemplos:

```
> (selecciona array-1 list)
((racket rkt) (prolog pl))
> (selecciona array-2 string?)
("suma" "lista vacia")
```

[2] e. o. c. = en otro caso

9. **`(colecta array funcion-aridad-1)`**: evalúa `funcion-aridad-1` con cada uno de los valores de `array`. Retorna una lista con los resultados de las evaluaciones.

Ejemplos:

```
> (colecta array-1 reverse)
((rkt racket) (pl prolog))
> (colecta array-3 procedure?)
(#t #t)
```

10. **`(detecta array predicado)`**: retorna un valor de `array` que al ser evaluado con `predicado` devolvió `#t`; e. o. c. `nil`.

Ejemplos:

```
> (detecta array-2 symbol?)
boolean
> (detecta array-1 procedure?)
nil
```

11. **`(compone array1 array2)`**: devuelve un array asociativo con los pares de `array1` y `array2`, a excepción de aquellos pares de `array2` cuya clave está en `array1`.

Ejemplos:

```
> (compone (array-asoc-crea) (array-asoc-crea))
(array-asociativo)
> (claves (compone array-1 array-3))
(8.6 5.07 "impar" "ultimo")
```

3. Ejemplo de implementación

Construcción de los arrays `array-1`, `array-2` y `array-3` (puede incluirlas en el programa junto a las funciones solicitadas).

```
(define array-1
  (agrega-reemplaza
    (agrega-reemplaza (array-asoc-crea) 8.6 '(racket rkt))
  5.07 '(prolog pl)))
```

```
(define array-2
  (agrega-reemplaza
    (agrega-reemplaza
      (agrega-reemplaza
        (agrega-reemplaza
          (agrega-reemplaza (array-asoc-crea) #\a 97)
        #t 'boolean)
      #f 'boolean)
    + "suma")
  '() "lista vacia"))


(define array-3
  (agrega-reemplaza
    (agrega-reemplaza (array-asoc-crea) "impar" odd?)
  "ultimo" (lambda (lista) (car (reverse lista)))))
```

Además de las definiciones anteriores, agregue otros ejemplos de arrays asociativos para plantear los casos de prueba, de manera que cubra todas las alternativas para las funciones solicitadas.

Ejemplo de una definición de función (no solicitada en la sección anterior) que retorna el array asociativo resultante de cambiar un par a partir de una clave dada. El nuevo valor del par (misma clave) es el resultado de aplicar la función del argumento al valor anterior.

```
(define aplica-a
  (lambda (array clave funcion-aridad-1)
    (if (not (incluye-clave? array clave)) array
      (agrega-reemplaza array clave
        (funcion-aridad-1 (valor-en array clave))))))
```