Lenguajes de Programación

Práctica 1 - Fundamentos de Racket

Semestre 2017-2

Fecha de inicio: 3 de febrero de 2017

Fecha de término: 17 de febrero de 2017

Instrucciones

- Completar de manera clara y ordenada las funciones del archivo `practica1.rkt`.
- Para tener derecho a calificación, la práctica debe ejecutarse sin errores ni advertencias. No está permitido utilizar primitivas de Racket que resuelvan directamente los ejercicios, ni modificar la firma de ninguna función.
- La entrega es en equipos de máximo 3 integrantes. Seguir los lineamientos especificados en: `http://lenguajesfc.com/lineamientos.html`.
- Los puntos extra son individuales y sólo se puede escoger uno. El punto extra teórico debe de realizarse **a computadora y entregarse impreso en hojas recicladas a más tardar el 17 de febrero durante la sesión de laboratorio**. El punto extra de programación se envía por correo electrónico en un archivo `<no.cuenta>_P01.rkt` enviar con el asunto `[LDP-EXP1]`. Incluir el nombre completo del autor en el cuerpo del correo.

Ejercicios

1. **Suma de potencias**

   Completar el cuerpo de la función `(pot-sum a b)` que toma dos números enteros y los eleva a sí mismos para luego sumar las potencias, es decir, debe regresar $a^b + b^a$. Ejemplos

```
> (pot-sum 1 7)
8
> (pot-sum 2 9)
593
> (pot-sum 8 3)
7073
```

2. **Valor absoluto**

   Completar el cuerpo de la función `(absoluto n)` que recibe un número y regresa su valor absoluto. Ejemplos:

```
> (absoluto 1)
1
> (absoluto -7)
7
> (absoluto 2)
2
```

3. **Cadenas de un lenguaje**

   Completar el cuerpo del predicado `(pertenece? s)` que recibe una cadena e indica si se encuentra en el lenguaje de todas las cadenas en $\{a, b\}$ tales que tienen dos letras `a` al inicio o al final.

   Ejemplos:

```
> (pertenece? aabbababababab)
#t
> (pertenece? bbabababababbbaba)
#f
> (pertenece? aabbbababababaabbababaa)
#t
```

4. **División por restas sucesivas**

   Completar la función `(division a b)` de forma recursiva para que permita hacer la división entera por restas sucesivas. Este método consiste en restar sucesivamente el divisor del dividendo hasta obtener un resultado menor que el divisor, que será el resto de la división; el número de restas efectuadas será el cociente. Ejemplos:

```
>(division 10 2)
5
>(division 7 2)
3
>(division 2 10)
0
```

5. **Limpiar cadenas**

   Completar la función recursiva `(limpia-cadena s)` para que dada una cadena, la "limpie", es decir, si hay caracteres adyacentes repetidos, debe dejar una única aparición. Ejemplos:

```
>(limpia-cadena ‘‘yyzzza’’)
‘‘yza’’
>(limpia-cadena ‘‘abbbcdda’’)
‘‘abcda’’
>(limpia-cadena ‘‘Hello’’)
‘‘Helo’’
```

6. **Números binarios**

   Completar la función `(par-impar s)` que recibe una cadena que representa un número binario para que responda si el número representado por la cadena es par o impar.

```
> (par-impar ‘‘111110110’’)
‘‘Par’’
> (par-impar ‘‘11011000001’’)
‘‘Impar’’
> (par-impar ‘‘11100101011)
‘‘Impar’’
```

7. **Funciones de orden superior**

Definir las siguientes funciones usando las primitivas `map`, `filter`, `foldr` o `foldl`:

a) Completar el cuerpo de la función `(binarios l)` que dada una lista de números, regresa otra con la representación binaria de cada uno de ellos. Ejemplos:

```
>(binarios '(1 2 3))
'(``1'' ``10'' ``11'')
>(binarios '(1 7 2 9))
'(``1'' ``111'' ``10'' ``1001'')
>(binarios '(0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15))
'("0" "1" "10" "11" "100" "101" "110" "111" "1000" "1001" "1010" "1011"
"1100" "1101" "1110" "1111")
```

b) Completar el cuerpo de la función `(longitudes l)` que recibe una lista de cadenas y regresa una lista de pares de la forma `(s, k)` donde `k` es la longitud de cada cadena. La lista final no debe tener elementos repetidos. Ejemplos:

```
> (longitudes '(``Hola'' ``Mundo''))
'((``Hola'' 4) (``Mundo'' 5))
> (longitudes '(``Lenguajes'' ``de'' ``Programación''))
'((``Lenguajes'' 9) (``de'' 2) (``Programación'' 12))
> (longitudes '(``Karla'' ``Ricardo'' ``Manuel''))
'((``Karla'' 5) (``Ricardo'' 7) (``Manuel'' 6))
```

c) Completar el cuerpo de las funciones `(reversar l)` y `(reversal l)` que devuelvan la reversa de una lista usando la función `foldr` y `foldl` respectivamente. Ejemplos:

```
>(reversar '(1 7 2 9))
'(9 2 7 1)
>(reversal '(1 7 2 9))
'(9 2 7 1)
```

8. **Operaciones sobre listas**

Redefinir o definir cada una de las siguientes operaciones sobre listas. No está permitido usar funciones de Racket que resuelvan directamente los ejercicios (`append, map, filter, take` o `drop`). Apoyarse de la técnica de cazamiento de patrones de ser necesario.

a) Completar el cuerpo de la función `(agrega-final l e)` que agrega el elemento `e` al final de la lista `l`.

b) Completar el cuerpo de la función `(promedio l)` que regresa el promedio de la lista `l`.

c) Completar el cuerpo de la función `(obten i l)` que regresa el `i`-ésimo elemento de la lista l.

d) Completar el cuerpo de la función `(n-primeros l)` que selecciona los primeros `n` elementos de la lista `l,` donde `n` es el elemento más pequeño de la lista. La lista debe ser de números.

e) Completar el cuerpo de la función `(unica-vez l)` toma una lista como parámetro y regresa otra lista con los elementos que aparecen una única vez en `l`.

9. **Expresiones `let` y lambdas**

   Definir las siguientes funciones combinando expresiones `let` y lambdas. Por ejemplo, para calcular el factorial de 10 puedes escribir:

```
(letrec ([fact (lambda (n)
             (if (< n 2) 1
                (* n (fact (- n 1)))))])
   (fact 10))
```

   a) Definir una función que regrese el mayor de los números 1834 y 1729.

   b) Definir una función recursiva que calcule la suma de los primeros 100 naturales.

Puntos extra

- (1.5 pts.) Leer en el artículo "*Why Functional Programming Matters*" de J. Hughes disponible en `http://worrydream.com/refs/Hughes-WhyFunctionalProgrammingMatters.pdf` y escribir una crítica de al menos dos cuartillas indicando aquellas secciones que consideren importantes de resaltar, aquellas con las que están de acuerdo, las ideas con las que sí y por qué.

- (1.5 pts.) Dada una lista de números enteros, definir una función que devuelve la mayor de las sumas de las sublistas de la misma. En este caso, las sublistas de una lista son todas las listas no vacías de elementos consecutivos de la lista original.

  Por ejemplo:

```
> (sublistas '(1 7 2 9))
((1) (7) (2) (9) (1 7) (7 2) (2 9) (1 7 2) (7 2 9) (1 7 2 9))
> (max-sumas '(1 7 2 9))
19
```