Lenguajes de Programación

Práctica 1

Semestre 2022-1
Facultad de Ciencias, UNAM

Profesora: Karla Ramírez Pulido
Ayud. Lab.: Silvia Díaz Gómez
Fhernanda Montserrat Romo Olea

Fecha de entrega: 08 de Octubre de 2021

Descripción

La práctica consiste en completar el cuerpo de las funciones faltantes del archivo **`practica1.rkt`**. No está permitido usar primitivas de Racket que resuelvan directamente los ejercicios.

Ejercicios

1. Completar el cuerpo de la función **`(area-lateral a b c)`** que dados el largo, ancho y altura de un paralelepípedo rectángulo respectivamente, calcula el área lateral del mismo. Fórmula:

$$Al = 2\,(a + b)\,c$$

```
;; area-lateral: number number number → number
(define (area-lateral a b c) ...)
```

2. Completar el cuerpo de la función **`(area-total g d)`** que dada la generatriz, la altura y el diámetro de la base de un cono circular recto, calcula el área total del mismo. Usar asignaciones locales **`let`** para evitar cálculos repetitivos. Fórmula:

$$At = \pi r g + \pi r^2$$

```
;; area-total: number number → number
(define (area-total g d) ...)
```

3. Completar el cuerpo del predicado **`(decremental? a b c d)`** que dados cuatro números indica si se encuentran ordenados de forma decremental. Por ejemplo:

```
(decremental? 1 7 2 9) => #f
(decremental? 1 8 3 5) => #f
(decremental? 6 3 1 0) => #t
```

```
;; decremental?: number number number number → boolean
(define (decremental? a b c d) ...)
```

4. Un niño quiere subir saltando una escalera. Con cada salto que da, puede subir 1, 2 o 3 escalones. Por ejemplo, si la escalera tiene tres escalones, la puede subir de cuatro formas distintas: 1, 1, 1; 1, 2; 2, 1 o 3. Completar el cuerpo de la función **`(numero-formas e)`** que dado el número de escalones de una escalera, indica el número de formas que puede subir saltando el niño. Por ejemplo:

```
(numero-formas 3)  => 4
(numero-formas 4)  => 7
(numero-formas 10) => 274

;; numero-formas: number → number
(define (numero-formas e) ...)
```

5. Se dice que un número natural es *raro* si al sumar cada una de sus cifras elevadas al número de cifras que lo forman, se obtiene el propio número. Por ejemplo, el número 153 (que tiene 3 cifras) es raro pues $153 = 1^3 + 5^3 + 3^3$. Completar el cuerpo del predicado **`(raro? n)`** que dado un número natural, indica si es raro. Por ejemplo:

```
(raro? 3)   => #t
(raro? 153) => #t
(raro? 12)  => #f

;; raro?: number → boolean
(define (raro? n) ...)
```

6. Completar el cuerpo de la función **`(rombo n)`** que dado un número, construya una cadena que dibuje un rombo con dicho número de dígitos. El número deberá estar entre 1 y 10. El rombo debe construirse con especificadores de formato como **`\n`** o **`\t`** y debe poderse mostrar con la función **`display`**. Por ejemplo:

```
(display (rombo 5)) =>

    0
   101
  21012
 3210123
432101234
 3210123
  21012
   101
    0

;; rombo: number → string
(define (rombo n) ...)
```

7. Completar el cuerpo de las siguientes funciones *recursivas* sobre listas.

   (a) Completar el cuerpo de la función recursiva **`(entierra s n)`** que dado un símbolo lo *entierra* **`n`** número de veces. Es decir, se deberán anidar $n-1$ listas hasta que se llegue a la lista que tiene como único elemento al símbolo correspondiente. Por ejemplo:

```
(entierra 'foo 5) => '(((((foo)))))

;; entierra: symbol number → list
(define (entierra s n) ...)
```

   (b) Una forma de encontrar los números primos en un determinado rango es mediante la conocida *Criba de Eratóstenes*. El algoritmo consiste en ir tomando los números del rango y eliminar todos los números que sean múltiplos de éste. El algoritmo terminará cuando no se pueda eliminar ningún número más.

   Por ejemplo, para encontrar los números primos del 2 al 20, se tiene la siguiente lista:

```
'(2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20)
```

   El primer paso consiste en eliminar todos aquellos números que sean múltiplos de dos (excepto el primero), con lo cual quedaría la siguiente lista:

```
'(2 3 5 7 9 11 13 15 17 19)
```

   Ahora, se pasa al siguiente número en la lista, en este caso el tres, y se repite el procedimiento:

```
'(2 3 5 7 11 13 17 19)
```

   Para el siguiente paso, se deben eliminar los múltiplos de cinco, sin embargo no queda ningún múltiplo de este número en la lista con lo cual, termina el algoritmo y se concluye que los números primos del 2 al 20 son: **`2`**, **`3`**, **`5`**, **`7`**, **`11`**, **`13`**, **`17`** y **`19`**.

   Completar el cuerpo de la función recursiva **`(criba-eratostenes n)`** que encuenta los números primos en un rango de 2 a n usando la *Criba de Eratóstenes*. Por ejemplo:

```
(criba-eratostenes 20) => '(2 3 5 7 11 13 17 19)

;; criba-eratostenes: number → (listof number)
(define (criba-eratostenes n) ...)
```

   (c) Un número puede representarse mediante el producto de números primos. Por ejemplo, el número 405 se puede representar como el producto de: $3^4 \times 5$.

   Para representar la descomposición en factores primos, suele usarse una tabla, a la izquierda se coloca el número a descomponer y a la derecha el número primo más pequeño por el que se puede dividir, en el siguiente renglón se coloca el resultado de la división del lado izquierdo y se busca el siguiente número primo a partir del número del renglón anterior. El proceso se detiene cuando la división resultante es uno.

405	3
135	3
45	3
15	3
5	5
1	

Al tener números repetidos en la columna derecha se pueden representar como potencias.

Completar el cuerpo de la función recursiva **(descomposicion-primos n)** que toma un número y regresa una lista de pares con la descomposición en factores primos del mismo. Por ejemplo:

```
(descomposicion-primos 405) => '((3 . 4), (5 . 1))
```

```
;; descomposicion-primos: number → (listof (pairof number))
(define (descomposicion-primos n) ...)
```

8. Completar los siguientes ejercicios haciendo uso de las funciones de orden superior **map**, **filter**, **foldr** y/o **foldl**. Para este ejercicio se prohíbe definir funciones auxiliares, en caso de requerirlas, usar funciones anónimas (**lambda**) en combinación de asignaciones locales **let**, **let*** o **letrec**.

   (a) Los números del 0 al 9 en japones se nombran de la siguiente manera:

0	→	*rei*
1	→	*ichi*
2	→	*ni*
3	→	*san*
4	→	*yon*
5	→	*go*
6	→	*roku*
7	→	*nana*
8	→	*haci*
9	→	*kyu*

El número 10 se nombra como *ju* y a partir de éste, pueden construirse los números del 11 al 99. Basta con indicar cuantas veces se suma el diez y cuantas unidades tiene. Algunos ejemplos:

- 20 se nombra *ni ju* pues se suma dos (*ni*) veces diez (*ju*).
- 37 se nombre *san ju nana* pues se suma tres (*san*) veces diez (*ju*) y se tienen siete (*nana*) unidades.
- 83 se nombra *haci ju san* pues se suma ocho (*haci*) veces diez (*ju*) y se tienen tres (*san*) unidades.

Completar el cuerpo de la función recursiva **(a-japones n)** que recibe una lista de números entre 0 y 99 regresa una lista con su representación en japones. Por ejemplo:

```
(a-japones '(20 37 83)) => '("ni ju" "san ju nana" "haci ju san")
```

```
;; a-japones: (listof number) → (listof string)
(define (a-japones n) ...)
```

(b) Completar el cuerpo de la función recursiva **(perfectos xs)** que recibe una lista de números y regresa una nueva lista que contiene únicamente aquellos que son perfectos. Un *número perfecto* es un número natural que es igual a la suma de sus divisores propios positivos. Por ejemplo, 6 es un número perfecto porque sus divisores propios son 1, 2 y 3; y $6 = 1 + 2 + 3$. Por ejemplo:

```
(perfectos '(1 6 2)) => '(6)

;; perfectos: (listof number) → (listof number)
(define (perfectos xs) ...)
```

(c) Completar, usando **foldr**, el cuerpo de la función recursiva **(aproxima n)** tal que:

$$aproxima\ n = \sum_{i=0}^{i=n} \frac{1}{i!}$$

Por ejemplo:

```
(aproxima 4) => 2.708

;; aproxima: number → number
(define (aproxima n) ...)
```

(d) Usando **foldl**, completar el cuerpo de la función recursiva **(rota xs)** que produce todas las rotaciones de una lista. Por ejemplo:

```
(rota '(1 2 3)) => '((1 2 3) (2 3 1) (3 1 2))

;; rota: (listof any) → (listof (listof any))
(define (rota xs) ...)
```

Requerimientos

Recuerda que la entrega de esta práctica es individual.

Deberás entregar tu práctica a través de la plataforma **Google classroom** antes de las 23:59 del día de entrega indicado, tal y como lo indican los lineamientos de entrega. Deberás adjuntar tu código en un archivo llamado **practica1.rkt**. Revisa bien mayúsculas, minúsculas y espacios para el nombre del archivo que se pide.

El orden en el que aparezcan las funciones en el archivo solicitado, debe ser el orden especificado en este PDF, de lo contrario podrán penalizarse algunos puntos. Puedes utilizar funciones auxiliares, se recomienda definirla después de la función que la va a utilizar. No dudes en aclarar tus dudas en la sesión de laboratorio y vía correo electrónico.

¡Que tengas éxito en tu primera práctica!.