Profesores: Federico Olmedo Ismael Figueroa
Auxiliares: Cristian Carrión Bastián Corrales Damián Arquez
Ayudantes: David Ibáñez Javier Kauer Martín Segur
Camilo Urzúa Daniela Assael Kathleen Kohler

Tarea 1
Programación Funcional en Racket

Para la resolución de la tarea recuerde que

- Toda función debe estar acompañada de su firma, una breve descripción coloquial (en `T1.rkt`) y un conjunto significativo de tests (en `test.rkt`).
- Todo datatype definido por el usuario (via `deftype`) debe estar acompañado de una breve descripción coloquial y de la gramática BNF que lo genera.

Si la función o el datatype no cumple con estas reglas, será ignorado.

Ejercicio 1 **60 Pt**

Una *fracción continúa* finita a coeficientes enteros es una expresión de alguna de las siguientes formas:

$$a_0, \quad a_0 + \frac{b_o}{a_1}, \quad a_0 + \frac{b_0}{a_1 + \frac{b_1}{a_2}}, \quad a_0 + \frac{b_0}{a_1 + \frac{b_1}{a_2 + \frac{b_2}{a_3}}}, \quad a_0 + \frac{b_0}{a_1 + \frac{b_1}{a_2 + \frac{b_2}{a_3 + \frac{b_3}{a_4}}}}, \quad \ldots$$

donde $a_i, b_i \in \mathbb{Z}$. La primera expresión es una fracción continua de grado 0, la segunda de grado 1, y así sucesivamente. Es fácil ver que una fracción continua es o bien una fracción continua "simple", de la forma $a_0 \in \mathbb{Z}$, o bien una fracción continua "compuesta", de la forma $a_i + \frac{b_i}{d}$ donde $a_i, b_i \in \mathbb{Z}$ y d es otra fracción continua. Por lo tanto, el conjunto $\mathsf{CFraction}$ de todas las fracciones continuas puede definirse *inductivamente*, a través del siguiente par de reglas:

$$\frac{a \in \mathbb{Z}}{(\mathsf{simple}\ a) \in \mathsf{CFraction}} \qquad \frac{a \in \mathbb{Z} \quad b \in \mathbb{Z} \quad d \in \mathsf{CFraction}}{(\mathsf{compound}\ a\ b\ d) \in \mathsf{CFraction}}$$

Por ejemplo, la fracción continua $3 + \frac{1}{4 + \frac{1}{12 + \frac{1}{4}}}$ se representa a través de

$$(\mathsf{compound}\ 3\ 1\ (\mathsf{compound}\ 4\ 1\ (\mathsf{compound}\ 12\ 1\ (\mathsf{simple}\ 4))))$$

(a) [6 Pt] Defina el tipo de datos recursivo `CFraction` y acompáñelo de su gramática.

(b) [5 Pt] Defina la función

```
;; eval :: CFraction -> Rational
```

que evalúa una fracción continua, devolviendo el número racional que representa.

(c) [5 Pt] Defina la función

```
;; degree :: CFraction -> Integer
```

que devuelve el grado de una fracción continua.

(d) [8 Pt] Defina la función

```
;; fold-cfraction :: (Integer -> A) (Integer Integer A -> A) -> (CFraction -> A)
```

que captura el esquema de recursión asociado a `CFraction`

(e) [8 Pt] Redefina las funciones `eval` y `degree` usando el `fold-cfraction` arriba definido (en vez de una recursión explícita). Si necesita definir una función como argumento del `fold-cfraction` , hágalo usando funciones anónimas.

(f) [10 Pt] Defina la función

```
;; mysterious-cf :: Integer -> CFraction
```

que genera la siguiente secuencia de fracciones continuas:

$$6, \quad 6+\frac{1^2}{6}, \quad 6+\frac{1^2}{6+\frac{3^2}{6}}, \quad 6+\frac{1^2}{6+\frac{3^2}{6+\frac{5^2}{6}}}, \quad 6+\frac{1^2}{6+\frac{3^2}{6+\frac{5^2}{6}}}, \quad 6+\frac{1^2}{6+\frac{3^2}{6+\frac{5^2}{6+\frac{7^2}{6}}}}, \quad \ldots \tag{1}$$

La primera fracción continua corresponde a `mysterious-cf 0`, la segunda corresponde a `mysterious-cf 1`, y así sucesivamente. Si a `mysterious-cf` se le pasa un argumento negativo, debe devolver un error, con el siguiente mensaje `''Error: argumento negativo''`.

Hint: Para encontrar el patrón y caso base, se le sugiere que escriba las fracciones de la Ecuación (1) utilizando notación de Racket, por ejemplo, `(simple 6)` y `(compound 6 (sqr 1)(simple 6))`, para las primeras 2.

(g) [8 Pt] Defina la función

```
;; mysterious-list :: Integer -> ListOf Float
```

que devuelve una lista tal que el i-ésimo elemento es calculado como la resta de la evaluación (utilizando `eval` o `eval2`) de `(mysterious-cf i)` menos 3. Luego de evaluar cada elemento, utilice la función `fl` para transformar los números fraccionarios a su representación en punto flotante.

Para implementar `mysterious-list` , parta definiendo una función

```
;; from-to :: Integer -> Integer -> ListOf Integer
```

que construye una lista de enteros comprendidos entre dos enteros dados, y luego defina `mysterious-list` en términos de `map` y `from-to`.

```
>>> (from-to 0 3)
'(0 1 2)
```

Viendo cuál es el resultado de `(mysterious-list n)` para distintos valores de `n`, responda la siguiente pregunta: ¿a qué número tiende `(mysterious-cf k)` cuando `k` tiende a infinito?

(h) [10 Pt] Por último, defina la función

```
;; rac-to-cf :: Rational -> CFraction
```

que transforma un número racional no-negativo en su representación en forma de fracción continua.

```
>>> (rac-to-cf (+ 3 49/200))
(compound 3 1 (compound 4 1 (compound 12 1 (simple 4))))
```

Para ello, siga el algoritmo descrito en este enlace.