Tarea 1
Programación Funcional en Racket

Para la resolución de la tarea, recuerde que

- La tarea es individual o de a dos. En caso de hacerla de a dos, debe entregarla sólo uno de los miembros del grupo, y recuerde incluir los nombres de ambos miembros en el archivo `base.rkt`. (Una vez entregada la tarea, no se permiten modificaciones en la composición del grupo.)
- En el material docente (`Clase 01.pdf`, diapositivas 7 y 8) se publicó la política de colaboración, reglamentando como está autorizado colaborar con compañer@s de otro grupo.
- Para resolver los distintos ejercicios, se permite definir funciones auxiliares. Éstas deben ir acompañada de su firma y una breve descripción coloquial.
- Todas las funciones presentes en el archivo `base.rkt` (auxiliares o ya incluidas) deben ir acompañadas de un conjunto "significativo" de tests en el archivo `test.rkt`.
- Las funciones que no lleven firma, descripción coloquial o tests no serán consideradas para la evaluación.
- De la misma forma, las funciones que no cumplan con las interfaces definidas en la tarea no serán consideradas para la evaluación.
- Los mensajes que se deban imprimir hacia el usuario (e.g. mensajes de error), deben ser idénticos a los especificados en enunciado de la tarea.
- La entrega vía U-Cursos debe constar de dos archivos: uno donde se encuentre el código fuente de la tarea (`base.rkt`) y otro con todos los tests que se usaron durante el desarrollo de ésta (`test.rkt`).
- Hay un plazo de dos semanas para la entrega. Se permiten hasta tres días de retraso, con una penalización de medio punto por día.

El objetivo de esta tarea es definir distintas operaciones sobre polinomios. Para representar polinomios vamos a usar el siguiente tipo inductivo:

```
#|
<Polynomial> ::= (nullp)
               |  (plus <Number> <Integer> <Polynomial>)
|#
(deftype Polynomial
  (nullp)
  (plus coef exp rem))
```

Por ejemplo, el polinomio $p(x) = 4x^5 + 3x^2 + 5$ se representa con el término `(plus 4 5 (plus 3 2 (plus 5 0 (nullp))))`. Observe que un mismo polinomio puede representarse de diferentes maneras. Por ejemplo, las siguientes son otras representaciones (equivalentes) de p: `(plus 3 2 (plus 4 5 (plus 5 0 (nullp))))`, `(plus 0 10 (plus 4 5 (plus 0 3 (plus 3 2 (plus 5 0 (nullp))))))`.

Ejercicio 1 (normalización) **20 Pt**

Decimos que la representación de un polinomio está en *forma normal* si i) los exponentes son listados de mayor a menor, estrictamente, y ii) los monomios de coeficiente 0 se omiten. Por ejemplo,

- `(plus 3 2 (plus 4 5 (plus 5 0 (nullp))))` y `(plus 4 5 (plus 7 5 (nullp)))` no están en forma normal porque violan la condición i);
- `(plus 0 10 (plus 4 5 (plus 0 3 (plus 3 2 (plus 5 0 (nullp))))))` no está en forma normal porque viola la condición ii);

(a) [7 Pt] Defina la función `nf?` :: `Polynomial -> Bool` que verifica si un polinomio está en forma normal.

```
> (nf? (plus 3 2 (plus 4 5 (plus 5 0 (nullp)))))
#f
> (nf? (plus 4 5 (plus 3 2 (plus 5 0 (nullp)))))
#t
```

(b) [13 Pt] Defina la función `normalize` :: `Polynomial -> Polynomial` que normaliza un polinomio.

```
> (normalize (plus 4 5 (plus 8 10 (plus 0 8 (plus 7 10 (plus 2 7 (nullp)))))))
(plus 15 10 (plus 2 7 (plus 4 5 (nullp))))
```

Hint: Para implementar la función `normalize` le puede resultar útil definir primero la función auxiliar `sumaMon` :: `Number Integer Polynomial -> Polynomial` que actúa de la siguiente manera: dado el polinomio `p` en forma normal, `sumaMon c m p` devuelve el polinomio en forma normal que resulta de sumarle a `p` el monomio $c\,x^m$. Por ejemplo,

```
> (sumaMon 6 6 (plus 4 4 (plus 2 2 (nullp))))
(plus 6 6 (plus 4 4 (plus 2 2 (nullp))))
> (sumaMon 3 3 (plus 4 4 (plus 2 2 (nullp))))
(plus 4 4 (plus 3 3 (plus 2 2 (nullp))))
> (sumaMon 10 2 (plus 4 4 (plus 2 2 (nullp))))
(plus 4 4 (plus 12 2 (nullp)))
> (sumaMon -2 2 (plus 4 4 (plus 2 2 (nullp))))
(plus 4 4 (nullp))
```

Ejercicio 2 (elementos básicos) **10 Pt**

(a) [5 Pt] Defina la función `degree` :: `Polynomial -> Integer` que devuelve el grado de un polinomio no nulo. En caso de recibir el polinomio nulo (`nullp`) como entrada, `degree` debe lanzar un error con el mensaje `"El polinomio nulo no tiene grado"`. Observe que la función debe admitir tanto polinomios normalizados como no normalizados.

```
> (degree (plus 4 4 (plus 5 5 (plus 1 1 (nullp)))))
5
```

Para ver el manejo de errores en Racket, consulte el PrePlai.

(b) [5 Pt] Defina la función `coefficient` :: `Integer Polynomial -> Number` que devuelve el coeficiente asociado a un exponente dado. Por ejemplo,

```
> ( coefficient  5 (plus 2 1 (plus 5 5 (plus 1 1 (nullp)))))
5
> ( coefficient  1 (plus 2 1 (plus 5 5 (plus 1 1 (nullp)))))
3
> ( coefficient  10 (plus 2 1 (plus 5 5 (plus 1 1 (nullp)))))
0
```

Como muestra el ejemplo, la función debe admitir tanto polinomios normalizados como no normalizados.

Ejercicio 3 (operaciones básicas) — 20 Pt

(a) [8 Pt] Defina la función `sumaPoly :: Polynomial Polynomial -> Polynomial` que suma dos polinomios (no necesariamente normalizados). La salida tampoco es necesario que esté normalizada.

(b) [12 Pt] Defina la función `multPoly :: Polynomial Polynomial -> Polynomial` que multiplica dos polinomios (no necesariamente normalizados). La salida tampoco es necesario que esté normalizada.

Hint: Para implementar la función `multPoly` le puede resultar útil definir primero la función auxiliar `mapPoly :: (Number Integer -> Number * Integer) Polynomial -> Polynomial` tal que `mapPoly f p` devuelve el polinomio que resulta de aplicar `f` a cada coeficiente y exponente de `p` . Por ejemplo, `(mapPoly (λ (c m) (cons (* c 2) (+ m 1))) p)` incrementa en uno todos los exponentes de `p` y duplica sus coeficientes. (*Hint:* ¿eso corresponde a hacer qué operación sobre `p`?)

```
> (mapPoly (λ (c m) (cons (* c 2) (+ m 1))) (plus 4 5 (plus 3 2 (plus 5 0 (
    nullp)))))
(plus 8 6 (plus 6 3 (plus 10 1 (nullp))))
```

Ejercicio 4 (fold) — 10 Pt

La función `foldPoly` debajo captura el esquema de recursión estructural sobre `Polynomial`:

```
;; foldPoly :: A (Number Integer -> A) -> (Polynomial -> A)
(define (foldPoly a f)
  (λ (p)
    (match p
      [(nullp) a]
      [(plus c g r) (f c g ((foldPoly a f) r))])))
```

Se pide que usando `foldPoly` defina la función `evalPoly :: Number -> (Polynomial -> Number)` que evalua un polinomio en un punto dado. Por ejemplo,

```
> ((evalPoly 3) (plus 2 3 (plus -6 2 (plus 2 1 (plus -1 0 (nullp))))))
5
```