FUNDAMENTOS DE COMPUTACIÓN
TRABAJO ENTREGABLE 2
NOVIEMBRE 2025

Este trabajo tiene un puntaje de 15 puntos. Puede ser realizado en grupos de hasta dos estudiantes, y se debe subir a Aulas antes del 27 de noviembre a las 21:00 hs.

Nota sobre el uso de inteligencia artificial:

El objetivo de este trabajo es evaluar los contenidos sobre recursión en listas y árboles, poniendo el foco en el razonamiento y la comprensión de los conceptos, más que en el resultado final. Las herramientas automáticas pueden usarse para revisar formato o sintaxis, pero **no para generar soluciones**. Los entregables que reproduzcan respuestas típicas de modelos de lenguaje (IA) serán revisados con especial atención y ante la duda se procederá a una defensa oral, o directamente a la pérdida de la totalidad de los puntos de la entrega.

1. Descripción general

Se desea implementar un entorno de cálculo con polinomios de una variable. Por entorno queremos decir un sistema que permita a un usuario ingresar una expresión polinómica y realizar cálculos con ella.
Las operaciones que se considerarán son suma y producto de polinomios de una misma variable con coeficientes enteros. Los polinomios pueden a su vez estar derivados o evaluados en un valor numérico entero.
Por ejemplo, una posible expresión polinómica es:

$$(3x^2 + 2x) * (6x^3 - 4)' + (x^4 - 8x^2 + 1)(2)$$

donde la coma ($'$) indica la operación de derivar un polinomio, y un número entre paréntesis luego de un polinomio (como por ejemplo el (2) luego de $(x^4 - 8x^2 + 1)$) indica la evaluación de dicho polinomio en el valor dado.

El resultado de esta expresión, teniendo en cuenta la precedencia de los operadores, es:

$$54x^4 + 36x^3 - 15$$

2. Polinomios

2.1. **Representación de los polinomios.**

Los polinomios se representarán como listas de monomios, donde cada monomio se representa como un par conteniendo el coeficiente y el exponente del mismo. Definimos los siguientes tipos en Haskell:

```
type Monomio = (Int,Int)
type Polinomio = [Monomio]
```

Así, el polinomio $3x^2 - 2x$ se representará como la lista `[(3,2),(-2,1)]`.

Esta representación tiene como inconveniente que un mismo polinomio pueda escribirse de muchas formas, ya que por ejemplo, todas estas son representaciones del mismo polinomio $3x^2 - 2x$:

- `[(-2,1),(0,1),(3,2)]`
- `[(1,4),(-1,1),(0,2),(3,2),(-1,4),(-1,1)]`
- `[(-5,1),(0,2),(1,2),(0,1),(3,1),(2,2)]`

Diremos que un polinomio está **reducido**, si no tiene monomios de grados repetidos o nulos, y está ordenado de mayor a menor según el exponente.

El polinomio `0` se representará con la lista vacía de monomios.

2.2. **Funciones básicas sobre polinomios.**

En todos los ejercicios se pueden utilizar las funciones del Preludio de Haskell definidas para los enteros: `(+), (*), (^), max, min`, etc., así como operadores de comparación `(==), (<), (>)`, etc. También se pueden utilizar las funciones sobre listas del Preludio vistas en clase: `sum, map, filter, (++)`, etc.

Se pide implementar las siguientes funciones sobre polinomios.

1) `agregarMon :: Monomio -> Polinomio -> Polinomio`, que recibe un monomio y un polinomio ya reducido, e inserta el monomio en su lugar en el polinomio, dejándolo reducido.
   Ejemplos: `agregarMon (1,2) [(4,6),(-3,2),(1,0)] = [(4,6),(-2,2),(1,0)]`
   `agregarMon (3,2) [(4,6),(-3,2),(1,0)] = [(4,6),(1,0)]`

2) `redPol :: Polinomio -> Polinomio`, que recibe un polinomio con sus monomios no necesariamente ordenados y posiblemente con monomios de grados repetidos o nulos, y lo reduce a su expresión mínima ordenándolo según el exponente de mayor a menor.
   Ejemplos: `redPol [(3,2),(4,6),(1,0),(-3,2),(5,0),(0,5)] = [(4,6),(6,0)]`
   `redPol [(1,4),(-1,1),(0,2),(3,2),(-1,4),(-1,1)] = [(3,2),(-2,1)]`
   `redPol [(-3,0),(0,7),(3,0)] = []`

Atención: Para las funciones que siguen se puede asumir que los polinomios que se reciben como argumentos ya están reducidos. Los resultados también deben estar reducidos:

3) `sumPol :: Polinomio -> Polinomio -> Polinomio`, que suma dos polinomios, devolviendo como resultado el polinomio reducido.
   Ejemplos: `sumPol [(2,3),(1,1)] [(3,1),(4,0)] == [(2,3),(4,1),(4,0)]`
   `sumPol [(2,3),(1,1),(-5,0)] [(3,4),(-2,3),(4,1)] = [(3,4),(5,1),(-5,0)]`

4) `mulPol :: Polinomio -> Polinomio -> Polinomio`, que multiplica dos polinomios, devolviendo como resultado el polinomio reducido.

Ejemplos: `mulPol [(2,3)] [(3,1),(4,0)] = [(6,4),(8,3)]`
`mulPol [(3,4),(-2,3),(4,1)] [(3,2),(2,0)] = [(9,6),(-6,5),(6,4),(8,3),(8,1)]`
`mulPol [] [(7,2),(-1,0)] = []`
Sugerencia: Defina funciones auxiliares que multipliquen monomios primero.

5) `derPol :: Polinomio -> Polinomio`, que calcula la derivada de un polinomio, devolviendo como resultado el polinomio reducido.
Ejemplos: `derPol [(3,2),(4,0)] = [(6,1)]`
`derPol [(7,0)] = []`
`derPol [(3,4),(-2,3),(4,1)] = [(12,3),(-6,2),(4,0)]`

6) `evalPol :: Polinomio -> Int -> Int`, que calcula el resultado de evaluar un polinomio en un valor entero dado.
Ejemplos: `evalPol [(3,2),(4,0)] 5 = 79`
`evalPol [(3,4),(-2,3),(4,1)] 0 = 0`

7) `gradoPol :: Polinomio -> Int`, que calcula el grado de un polinomio, esto es, el mayor exponente con coeficiente no nulo.
Ejemplos: `gradoPol [(3,2),(4,0)] = 2`
`gradoPol [] = 0`

3. Show

Defina las funciones que permiten mostrar a los polinomios de la de forma usual.
Sugerencia: utilizar la función `show` del Preludio de Haskell, que devuelve la representación de los elementos de los tipos estándar como Strings. Por ejemplo `show 35 = "35"`.

8) `showMon :: Monomio -> String`, que devuelve el string con la forma en que se escribe normalmente un monomio. Para indicar potencia se deberá utilizar el caracter `'^'`, de modo tal que x^k se representará como el string `"x^k"`.
Ejemplos: `showMon (3,2) = "3x^2".`
`showMon (1,5) = "x^5"`
`showMon (0,3) = ""`
`showMon (-7,4) = "-7x^4"`
`showMon (-2,0) = "-2"`

9) `showPol :: Polinomio -> String`, que devuelve el string con la forma en que se escribe normalmente un polinomio, es decir como sumas de monomios.
Ejemplos: `showPol [(1,4),(3,2),(-2,1),(7,0)] = "x^4 + 3x^2 - 2x + 7"`
`showPol [] = ""`
No es necesario poner espacios entre los operandos, el resultado del primer ejemplo puede ser también `"x^4+3x^2-2x+7"`

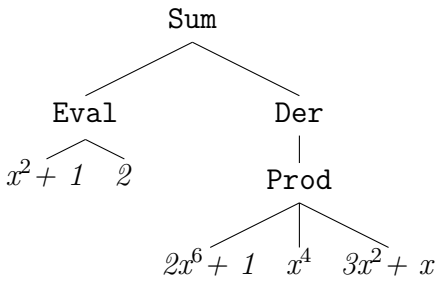
Importante: En ambas funciones se deberá tener en cuenta casos particulares, como por ejemplo cuando el exponente es 0 o 1, o el coeficiente es 0, 1 o negativo.

4. Expresiones polinómicas

Las expresiones polinómicas que se considerarán se obtienen mediante la suma y el producto de polinomios. Estos polinomios que se suman y multiplican pueden a su vez estar derivados o evaluados en un valor numérico entero. Las expresiones polinómicas se representarán mediante árboles. Para ello, se define el siguiente tipo de datos:

```
data ExPol where {Pol :: Polinomio -> ExPol ;
                  Der :: ExPol -> ExPol ;
                  Eval :: ExPol -> Int -> ExPol ;
                  Sum :: [ExPol] -> ExPol ;
                  Prod :: [ExPol] -> ExPol }
```

Por ejemplo para la expresión polinómica: $(x^2+1)(2)+((2x^6+1)*(x^4)*(3x^2+x))'$, el árbol correspondiente es:



y la expresión Haskell que lo representa es:

```
e :: ExPol
e = Sum [Eval(Pol[(1,2),(1,0)]) 2,
         Der (Prod [Pol[(2,6),(1,0)],
                    Pol[(1,4)],
                    Pol[(3,2),(1,1)]
                   ]
              )
          ]
```

Defina las siguientes funciones para operar con expresiones polinómicas

10) `cantPol :: ExPol -> Int`, que calcula la cantidad de polinomios que hay en las hojas de una expresión polinómica (sin evaluarla).
Para la expresión del ejemplo, la función `cantPol` devuelve `4`.

11) `cantx :: ExPol -> Int`, que calcula la cantidad de ocurrencias de la indeterminada (`"x"`) en una expresión polinómica (sin evaluarla).
En la expresión del ejemplo hay `5`: $(x^2+1)(2)+((2x^6+1)*(x^4)*(3x^2+x))'$

12) `maxProd :: ExPol -> Int`, que calcula el largo del producto más largo que hay en una expresión polinómica (sin evaluarla).
Para la expresión del ejemplo, la función `maxProd` devuelve `3`.

13) `gradoEP :: ExPol -> Int`, que calcula el grado (máximo exponente) que aparece en una expresión polinómica (sin evaluarla).
Para la expresión del ejemplo el máximo grado es `6`: $(x^2+1)(2)+((2x^6+1)*(x^4)*(3x^2+x))'$

14) `calcEP :: ExPol -> Polinomio`, que recibe una expresión polinómica y calcula el polinomio resultado de la misma (el cual deberá estar reducido).
Para la expresión del ejemplo, `calcEP` devuelve `[(72,11),(22,10),(18,5),(5,4),(5,0)]`, ya que:
   - $2^2+1=5$
   - $((2x^6+1)*(x^4)*(3x^2+x))=6x^{12}+2x^{11}+3x^6+x^5$
   - $(6x^{12}+2x^{11}+3x^6+x^5)'=72x^{11}+22x^{10}+18x^5+5x^4$
   - $5+(72x^{11}+22x^{10}+18x^5+5x^4)=72x^{11}+22x^{10}+18x^5+5x^4+5$

15) `resultado :: ExPol -> String`, que recibe una expresión polinómica y devuelve un String que representa el polinomio reducido resultado de la misma.
Para la expresión del ejemplo, `resultado` devuelve el siguiente String:
`"72x^11 + 22x^10 + 18x^5 + 5x^4 + 5"`

5. Entregables

- El trabajo deberá realizarse en grupos de hasta dos estudiantes.
- La entrega deberá realizarse por Aulas antes del **27/11/2025 a las 21:00 hs.**
- Deberán subirse a Aulas dos archivos (`Polinomios.hs` y `ExPol.hs`) con el código fuente de la solución.
- Deberá hacerse una única entrega por equipo. El archivo deberá incluir, como comentarios al inicio, los nombres y números de estudiante de cada integrante.
- En Aulas se encuentran los archivos `Polinomios.hs` y `ExPol.hs` con las funciones que deben implementarse. Para facilitar la corrección deberán usarse como templates para la entrega.
- No se corregirán archivos que no compilen, por lo que recomendamos comentar el código que no compile y dejar explícitamente como `undefined` las funciones no implementadas.
- Se utilizarán herramientas para la detección de copias y de uso de herramientas de IA, por lo que recomendamos NO UTILIZAR HERRAMIENTAS DE GENERACIÓN AUTOMÁTICA, NI COMPARTIR CÓDIGO.
- Para comprobar autoría, este trabajo entregable tendrá una instancia de defensa durante el 2º parcial.