TECNOLOGÍA DE PROGRAMACIÓN

Guión de la práctica 5

1.- Objetivo de la práctica

Los objetivos de esta práctica son:

- Introducción al uso de Haskell.
- Manejo de tipos de datos incluidos en Haskell: listas y tuplas.
- Comprensión de la recursividad funcional y de funciones de orden superior.

2.- Representación de polinomios

En esta práctica, vas a programar en Haskell una serie de funciones que trabajan sobre **polinomios de una sola variable**: https://es.wikipedia.org/wiki/Polinomio

Hay múltiples representaciones internas de polinomios, también dependientes del lenguaje de programación. En esta práctica vas a trabajar sobre dos representaciones internas en Haskell. Dado un polinomio como el siguiente:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0 x^0$$

las dos representaciones sobre las que trabajarás son:

- **Vector de coeficientes:** En una lista de números reales, el polinomio anterior quedaría representado como

  `[an, an-1, … , a1,a0]`

- **Lista de tuplas:** Cada tupla representaría un monomio con el coeficiente real y el exponente entero. El polinomio anterior quedaría representado como

  `[(an,n),(an-1,n-1), … (a1, 1), (a0, 0)]`

Las **diferencias principales** entre el vector de coeficientes y la lista de tuplas son que la lista de tuplas ocupa más espacio por cada elemento (a fin de cuentas, guardamos también el coeficiente). A cambio, la lista de tuplas no necesita almacenar los elementos cuyo coeficiente es 0, mientras que en el vector de coeficientes, es obligatorio ya que el coeficiente se deduce de la posición en el vector. Por ejemplo el polinomio $3x^{100}$ requeriría 101 elementos como vector de coeficientes pero solo uno como lista de tuplas.

Deberás crear dos módulos: `ArrayPolynomial.hs` y `TupleListPolynomial.hs`, que contengan ambas representaciones. Para que el modulo sea correcto, la primera línea de `ArrayPolynomial.hs` deberá ser

```
module ArrayPolynomial where
```

(con una primera línea equivalente para `TupleListPolynomial.hs`).

En cada uno de esos módulos, deberás implementar según la representación correspondiente las funciones que trabajan sobre polinomios que te proponemos en el siguiente apartado.

3.- Funciones

Vamos a intentar abstraer en la medida de lo posible la representación interna del polinomio de las funciones a través de las cuales se accede a dicho polinomio. Para cada una de las dos representaciones (dentro de cada uno de los respectivos módulos) **se deberán implementar las siguientes funciones**:

- `x` – Esta función devolverá un polinomio que sólo contiene una *x* (el coeficiente de grado 1 vale 1).
- `coef c` – Esta función devolverá un polinomio que contenga como término independiente la constante *c* (el coeficiente de grado 0 vale *c*).
- `padd lp` – Esta función suma todos los polinomios que haya en la lista lp y devuelve el resultado.
- `pmul lp` – Esta función multiplica todos los polinomios que haya en la lista lp y devuelve el resultado.
- `peval p x` – Esta función evalúa el polinomio p para el valor x, devolviendo como resultado el correspondiente número real resultado.
- `pderv p` – Esta función calcula la derivada analítica del polinomio p, dando como resultado otro polinomio.

Puedes implementarlas como quieras con las herramientas de las que dispongas. Se valorará:

- El uso correcto de la recursividad, cuando proceda.
- El uso de funciones de orden superior de Haskell (por ejemplo, map, foldr...) cuando proceda. Se deberá evitar la "reimplementación" de dichas funciones mediante recursividad.
- La eficiencia: deberás intentar que tus funciones sean razonablemente eficientes. Te damos los siguientes consejos:
  - La función `length` no se calcula instantáneamente, así que intenta evitar usarla una y otra vez (puede que este consejo te sea útil en la representación de polinomio como vector).
  - Aunque no es requisito fundamental que la representación mediante lista de tuplas esté ordenada, ciertas operaciones puede que sean más eficientes si sí que lo están, y puede que salga "rentable" asegurarse de que las tuplas siguen ordenadas.
- Las funciones no deberán ser genéricas, sino que dependerán de tipos de dato concretos.

4.- Pruebas

Te proporcionamos el archivo `PolynomialTest.hs` para que pruebes las funciones que has implementado anteriormente para las dos representaciones que te hemos propuesto. Úsalo como inspiración, pero haz tus propias pruebas, con diferentes polinomios de diferentes grados. Por supuesto también puedes utilizar el intérprete de Haskell para hacer pruebas. Sin embargo, deberás de asegurar que el archivo `PolynomialTest.hs` que te proporcionamos permite compilar, importando el módulo correspondiente, tus dos versiones de la implementación de polinomios. En caso de no compilar la calificación de la práctica será de 0.

5.- Para entregar

Como resultado de esta práctica deberás entregar los siguientes archivos:

- `ArrayPolynomial.hs` – que contenga la representación de polinomios en forma de vector de coeficientes. Todas las funciones deben tener **exactamente el mismo nombre** expresado en este guión.
- `TupleListPolynomial.hs` – que contenga la representación de polinomios en forma de lista de tuplas. Todas las funciones deben tener **exactamente el mismo nombre** expresado en este guión.

Deberás incluir los archivos que te pedimos en un archivo comprimido con el siguiente nombre:

- `practica5_<NIP>.zip` si la práctica se realiza de forma individual (donde <NIP> representa los 6 dígitos del NIP del alumno).
- `practica5_<NIP1>_<NIP2>.zip` si la práctica se realiza por parejas (donde <NIP1> y <NIP2> representa los 6 dígitos del NIP de cada alumno de la pareja, respectivamente).

Las dos representaciones de polinomios deberán compilar con el archivo de ejemplo `PolynomialTest.hs`. En caso de no hacerlo la calificación de la práctica será de 0.