Representation

A polynomial is the sum of several monomials. We chose to represent our polynomials as a list of a custom data type `Monomial`. These **Monomials** contain both a coefficient (integer) and a literal part, which we called **Symbols**. `Symbols` is another list, this time of **Symbol**. Finally, `Symbol` is a variable (character) powered to some exponent (integer),for example: x^3 -> Symbol 'x' 3 -> Char 'x', Exponent 3, where `Exponent` is a custom type equal to Int

Strategy

a) We first normalize the monomials. We sort their symbols (lower char), so that the same chars appear together and then transverse the list, fusing the symbols representing the same variable (and adding up the exponents). Afterward, we sort the polynomial itself (bigger degree -> lower char in symbol. This means that monomials with the same literal will appear together. Afterwards we transverse the list and fuse any monomials with the same literal (adding up the coefficients.

b) The sum of two polynomials is still a polynomial. All that is needed is to concatenate the inputs and then normalize them (as done in a)

c) We first use the distributive property twice: once to sum the product of a polynomial with each of the components of the other and afterwards to calculate each addend in a similar manner (we divide the polynomial into monomials and multiply them with the other monomial); ending up with a sum of monomials. Now, just like in b, we normalize.

d) We calculate the derivative of each monomial (a simple function based on the exponent of the chosen variable) and then normalize.

Usage

We have a straight-forward interface, which provides instructions regarding how to use it. You can use `0*x^2 + 2*y + 5*z^1 - 7*y^2 + x - 54*x*y^2*z^3`, a long but still normal polynomial; `2xx^2 + 14*y + 3z + x*yzx^2 + 3y + 0z + 8x^3`, a more badly-written and confusing one or even something like `tambem pode usar uma frase` to test our project.

To use the interface in ghci run:

```
stack ghci

ghci> :load Interface.hs

ghci> main
```

The coefficient and the exponent should be in one piece, without operations within itself (2*3*x) and in its proper place (before the literal or after an '^'). Other than that, you can write whatever you want.

You can also use the functions instead of the Interface:

Make sure you have loaded the `Polynomial.hs` file in ghci

Normalize Example

```
ghci> polynomialToString (normalize (parsePolynomial "0*x^2 + 2*y + 5*z^1 -
7*y^2"))

"-7y^2+2y^1+5z^1"
```

Sum Example

```
ghci> polynomialToString (sumPolynomial (parsePolynomial "0*x^2 + 2*y + 5*z^1 -
7*y^2") (parsePolynomial "3y^2 - x"))

"-4y^2-1x^1+2y^1+5z^1"
```

Multiplication Example

```
ghci> polynomialToString (multiplyPolynomial (parsePolynomial "0*x^2 + 2*y + 5*z^1
- 7*y^2") (parsePolynomial "-x"))

"7x^1y^2-2x^1y^1-5x^1z^1"
```

Derivative Example

```
ghci> polynomialToString (poliDerivative (parsePolynomial "0*x^2 + 2*y + 5*z^1 -
7*y^2") 'y')

"-14y^1+2"
```

We have created tests in `Tests.hs`, to run them, make sure you have installed `QuickCheck` and run them in ghci:

```
ghci> runTests

=== prop_normalizeExtraZeroValue from Tests.hs:29 ===
+++ OK, passed 100 tests.
...
...
```

These tests may take a while.