PFL_TP1

Polynomial Manipulation

Developed by: - Rui Pires up202008252@up.pt - Guilherme Moreira up202007036@up.pt

Description

This project consists of a program that manipulates polynomials. The program is able to read polynomials via user input and perform operations on them. The operations that can be performed are: - Normalization - Addition - Multiplication - Derivation

Internal Representation

We chose a representation of polynomials as a list of tuples, where each tuple is a pair of the form (coefficient,list of exponents). The list of exponents is a list of tuples, where each tuple is a pair of the form (variable,exponent). So to sum it up the internal representation is a list of tuples of the form (coefficient,list of tuples of the form (variable,exponent)). For example, the polynomial 2xy^2 + 3x^2y^3 would be represented as:

```
[(2,[(x,1),(y,2)]),(3,[(x,2),(y,3)])]
```

We chose this representation because it was the most intuitive and easy to implement. It also allows for easy manipulation of the polynomials, since we can easily add, multiply, normalize and derive them.

Normalization

The normalization of a polynomial is done in 6 important steps: Firstly, we use a function to remove all unwanted characters (in this case, spaces and '*') and insert each monomial into a list. This function is called **polynomialOrganizer**. After that, we use a function to convert a list of monomials into the internal representation we use. This function is called **internalRepresentation**. Next, we use a function called **polynomialSorter** to sort our the polynomial by their degree. Then, we call a function to add all the coefficients of monomials with the same set of variables. This function is called **sorting**. Next, we get to the part of reverting the polynomial from the internal representation form to a string. For this, we use a function for each of the monomials in a list which will convert them from an internal representation to a string representation. This function is called **tplToString**. Finally, we use a function called **joiner** to join all the monomials from the list of strings into a single string, to be outputted in a way the user can understand. The function that adds all of these functions together is called **normalize**.

Addition

The addition of two polynomials is done in the exact same way as the normalization, except for the first step. After the user provides the two polynomials to be added, we concatenate the two strings, with a '+' in the middle of them. After that, we just call the **normalize** function and it will provide de addition of both polynomials. The function that adds all of these functions together is called **add**.

Multiplication

The multiplication of two polynomials is done in 3 important steps: First, we use a function capable of multiplying a list of variables by a list of variables, which is done by appending the variables of the second list to the first one. This function is called **multiplyVars**. Second, we use a function capable of multiplying a monomial by another monomial which is done by multiplying the coefficients and multiplying the variables, using the multiplyVars function. This function is called **multiplyOne**. Finally, we use a function capable of multiplying a polynomial by another polynomial, which is done by multiplying each monomial of the first polynomial by each monomial of the second polynomial, using the multiplyOne function. This function is called **multiply**. We use a function called **simplifyExponents** to simplify the exponents of the variables in the polynomial (e.g. 2x^2x^3 = 2x^5). After that, we simply transform the internal representation of the polynomial into a string with the function multiplication. The function that adds all of these functions together is called **multiplication**.

Derivation

The derivation of a polynomial has the normal case, for most strings, and a special case. Starting off with the special case, this will happen if the string given to be derived is a single digit, with no variables. Whatever the variable to de derived is, it will always be 0 because the derivation of a constant is always 0. Moving on to the main part of the function: We start off the same way as the normalization, separating the string into a list of monomials in the internal representation form. From there, we use a function called **changer** which has the objective of changing the internal representation from the monomials to the internal representation of those same monomials after being derived. To do that, we use two functions inside a list comprehension: **exponentt** and **reducer**. The **exponentt** function is used to find the exponent of the variable to be derived (e.g coefficient x^2 = 2). The **reducer** function is used to reduce the exponent of monomials that have the variable to be derived and multiply it by the coefficient of the monomial. After that, the same happens as it does in the **normalize** function, we add monomials with the same set of variables, convert it from a list of internal representation to a list of strings and then concatenate them all together, to have the final result as a single string. The function that adds all of these functions together is called **derivative**.

Examples of use

Firstly, we load ghci with the file "trabalho.hs". After that, we call the main function by writing "main" in the terminal. We will then be provided with this menu:

You should then insert 1, 2, 3 or 4 and press Enter.

- If you choose 1, you will write down the polynomial you want to normalize and press Enter. (e.g. 2xy *Enter*)
- If you choose 2, you will write down the first polynomial you want to add, press Enter, write down the second polynomial you want to add and press Enter once again. (e.g. 2xy *Enter* 2xy *Enter*)
- If you choose 3, you will write down the first polynomial you want to multiply, press Enter, write down the second polynomial you want to multiply and press Enter once again. (e.g. 2xy *Enter* 2xy *Enter*)
- If you choose 4, you will write down the polynomial you want to derive, a space and the variable you want to differentiate by, then press Enter. (e.g. 2xy x)