Practical work PFL 1

Team

- Lia Vieira - up202005042
- Miguel Teixeira - up202005208

Internal representation of polynomials

Term = (n,[(v,e)]) Pol = [Term]

A polynomial can be represented in different formats. For us, a polynomial is an array of Terms and a Term is a tuple. The first part of each Term is the coeficient (n), which can be any positive or negative integer, and the second part is another array of tuples. This time, the first part of the tuple is a Char (v), which represents the variable, (eg: 'x'), and the second part represents the degree of that variable. We chose this representation for several reasons:

- We assumed that a Term is a simple monomial so it has a number and the following variables with their degrees.
- A number n can be followed by more than one variable so we needed a way of storing multiple variables, an array. In the end, the Term "2x^2^y^3^" becomes (2,[(x,2),(y,3)]) in our representation.
- Finally a polynomial is in fact a set of Terms, so the polynomial "3x^2^+2x^3^-y^2^", becomes [(3,[(x,2)]), (2,[(x,3)]), (-1, [(y,2)])].

Functionalities

Normalization

The process of normalizing consists of iterating throught a polynomial and add the terms with the same variables and simplifying each term when they have two variables with the same literal. Lastly, the polynomial is sorted by alphabetical order and then by degree. To normalize a polynomial the function 'normalizeP' must be called followed by a polynomial in a string format.

```
 normalize:: Pol -> Pol
normalize a = iter2 (length (simplifyPol a)) (simplifyPol a)
```

```
 *Main> normalizeP "2x^2 + 3x^2"
"5x^2"
*Main> normalizeP "y^2 + 3z^2z^3 + 2y^2"
"3z^5 + 3y^2"
```

Sum

To sum two polynomials, the lists are concatenated and the result is normalized. If any term ends with a coeficient 0 it is elimitaned. To sum, the function 'sumP' must be called followed by two polynomials in a string format.

```
 addPol::Pol->Pol->Pol
addPol a [(b,[])] = cleanPol(a ++ [(b,[])])
addPol a b = cleanPol (normalize (a++b))
```

```
 *Main> sumP "2x^2" "3x+4y+4x^2"
"6x^2 + 3x^1 + 4y^1"
*Main> sumP "2x^2" "0"
"2x^2"
*Main> sumP "3" "3x + 4y + 4x^2"
"4x^2 + 3x^1 + 4y^1 + 3"
```

Multiplication

The multiplication consists of multiplying every term of the first polynomial with every term of the second one. The fuction receives two polynomials. If one of them is empty, the fuction returns the other one but if one of them is "0" the fuction returns "0". To multiply, the function 'multP' must be called followed by two polynomials in a string format.

```
 multPols::Pol->Pol->Pol
multPols [] pol2 = pol2
multPols [(0,_)] pol2 = []
multPols pol1 pol2 = cleanPol (normalize [multTerms x y | x <- pol1, y <-pol2])
```

```
 *Main> multP "2x^2" "3x+4y+4x^2"
"8x^4 + 6x^3 + 8x^2y^1"
*Main> multP "2x^2" "0"
""
*Main> multP "3" "2x + 4y + 4x^2"
"12x^2 + 6x^1 + 12y^1"
```

Derivation

To calculate the partial derivative two arguments must be given: the variable by which you want to derive and a polynomial. Our fuction traverses the polynomial and checks if it has the variable that what given in the first argument. If it does, it reduces the degree of the term and multiplies it with the coeficent. If the variable is not in the term, that term is eliminated. To derive a polynomial the function 'deriveP' must be called followed by a char and a polynomial in a string format.

```
 diffPol::Char->Pol->Pol
diffPol c [] = []
diffPol c (x:xs) = cleanPol (normalize (([diffTerm c x]) ++ diffPol c xs))



exediffTerm::Char->Term->Term
exediffTerm c (n,((v,y):xs)) | c == v = (n*y,(v,y-1):xs)
                             | otherwise = addExp (v,y) (exediffTerm c (n,xs))
```

```
 *Main> deriveP 'x' "3x+4y+4x^2"
"8x^1 + 3"
*Main> deriveP 'y' "2x^2 + 3z"
""
*Main> deriveP 'x' "4xy + 2x^" + 4x^2"
"12x^1 + 4y^1"
```

Strings as input

Our program receives strings as input and converts them in our internal representation. To do that, we first split the string using '+' or '-' as delimiter and remove all the spaces. After that, we create the monomials and convert them to our representation and put everything together.

```
 createPol :: String -> Pol
createPol x = [z | z <- pol]
            where pol = [createCoeficient m | m <- createMono (removeSpaces (split x))]
```

Strings as output

To convert a polynomial to string we traverse it and convert every term to string placing a '^' between the literal and the degree and then place a '+' between every term. If it's negative it places a '-' instead of a '+'.

```
 outputTerm:: [(Char, Int)] -> String
outputTerm [] = []
outputTerm [(a,b)] = a:'^':(show b)
outputTerm ((a,b):xs) = a:'^':((show b)++(outputTerm xs))
```

```
 *Main> outputPol [(2,[('x',3),('y',3)]),(4,[('z',2)])]
"2x^3y^3 + 4z^2"
```

Testing

The file test.hs contains some examples of polynomials and the output when our functions are called.