Polyfun.hs

ReadMe

Group ?
UP 202202993 - Remus Comeaga
UP 202006950 - Vicente Salvador Martinez Lora

IMPORTANT NOTE: We didn't use the Data.Char library, as it was only later announced that we could use it.

Structure

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

1. Internal Representation

(Short description and justification of the choices of internal representation of Polynomials)

Variable

```
type Var = String
```

The Variable was represented as a type synonym of the type String, as it is normally composed of an alphabetical character or a compilation of alphabetical characters.

Coefficient

```
type Coef = Int
```

The Coefficient was represented as a type synonym of the type Int, because it's a number and we wanted the app to only work with simpler polynomials than ones where the coefficient is not an integer.

Exponent

```
type Exp = Int
```

The Exponent was represented as a type synonym of the type Int, because it's a number and we wanted the app to only work with simpler polynomials than ones where the exponent is not an integral.

Components

Components were not formally declared, but we thought of a component as the fragments that compose a monomial, which can be: Variables, the Coefficient, and the Exponents attached to each variable. This was to facilitate the creation of the monomials from the strings, as we searched for each component that we could find and joined together into a monomial.

Monomial

```
type Monomial = ([Var], Coef, [Exp])
```

The monomial was represented as a tuple with three items of each possible component type. The first item is the list of variables that compose the monomial, the second item is the coefficient of the monomial and the third item is the list of exponents present in the monomial.

We chose to represent both the variables and exponents in lists, as we could match each variable to its exponent by the index they're at. In other words, for the monomial "x^1 * y^2", the resulting list of variables would be ["x", "y"] and the list of exponents would be [1, 2].

So if we wanted to get the first variable and its exponent, we just needed to call the first item of each list with index 0.

In haskell
For the variable:
["x", "y"] !! 0
For the exponent:
[1, 2] !! 0

Giving the following outputs correspondingly, **"x"** and **1**.

Polynomials

```
type Polynomial = [Monomial]
```

In mathematics the polynomial is just a collection of monomials added to each other. So we chose to represent the polynomial as a list of monomials. This was to facilitate how the operations were done on polynomials, as we could always fragment each into its monomials and do the operations from a lower level of complexity.

2. Functionalities Implementation

(A short description of the strategy of implementation of each functionality)

Parsing

For parsing the string inputs we did the following steps: (Using examples)

1 - Separating the Monomials

	"0*x^2 + 2*y - 5*z + y + 7*y^2"
1. Remove any spaces:	"0*x^2+2*y-5*z+y+7*y^2"
2. Add plus signs before every minus sign	"0*x^2+2*y**+**-5*z+y+7*y^2"
3. Replace every plus sign with a space	"0*x^2 2*y -5*z y 7*y^2"
4. Split the string in every space	["0*x^2", "2*y", "-5*z", "y", "7*y^2"]

2 - Converting the monomial to internal form through the components

	"0*x^2"
1. Separating the components (splitting on the "*") ["0","x^2"]	
2. Finding the components that are variables and their exponents	(["x"], [2])
3. Finding the components that are coefficients	0
4. Joining everything	(["x"], 0, [2])

3 - Converting the polynomial to internal form

1. Joining all the monomials

[(["x"], 0, [2]), (["y"], 2, [1]), (["z"], -5, [1]), (["y"], 1, [1]), (["y"], 7, [2])]

Extra Notes:

- It can consider variables with more than 1 character and variables that include numeric values such as x1 or "x_1".
- It's mandatory to express "*" when multiplying

Normalization

1. Removing any monomials with coefficient 0
2. Sorting variables in each monomial alphabetically : "x^2*y*x" becomes "x^2*x*y"
3. Multiplying variables with the same base in each monomial : "x^2*x*y" becomes "x^3*y"
4. Sorting the polynomial alphabetically : "y+z+x" becomes "x+y+z"
5. Adding together joinable monomials : "5*x^2 + x + y^2" becomes "6*x^2 + y^2"
6. Sorting polynomial based on exponents: "x^2 + x^4" becomes "x^4 + x^2
7. Removing any monomial with coefficient 0 or any unnecessary variable with exponent equal to 0, that could have resulted from the actions before.

Addition

Addition is done by concatenating the polynomials and normalizing the output. As the normalization already deals with adding joinable monomials, we just create a polynomial composed of the monomials of all the polynomials to be added.

Multiplication

Multiplication of two polynomials is done by forming groups of two monomials, each monomial from polynomial 1 with each monomial from polynomial 2.

Each group is then processed into a single monomial by concatenating the variables and the exponents together, then multiplying the coefficients. The normalizer is then applied to the output, as normalization already deals with multiplication of variables with the same base.

Derivation

The derivation of the polynomial starts by finding the derivation of each monomial, as **the derivative of a sum is equal to the sum of the derivatives,** and a polynomial is just the addition of monomials.

In each monomial, we check if it's composed of just a coefficient, as the derivative of a constant is always equal to 0. If it's not, then we check if the variable we're referring to is present on the expression. In the case it's absent, we return the exact same monomial, else we apply the formula f'(x) = a*n*x^(n-1).

GHCI - Framework

We chose to include a ghci version of our application, where the user can use all the essential functions needed to complete the requested operations (normalization, addition, multiplication, derivation) and more. To run this version please follow the steps given in the page X of the Testing topic. Furthermore, in the file "framework.hs" there is also further information on how to run it and a description of every single function included.

GHC/.exe

We chose to give the user a simple interface where they can pick which operation they want to do by entering one of the characters given on the prompt: 1 for normalization, 2 for addition, 3 for multiplication, 4 for derivation, "Q" (or "q") - for quitting the application. Depending on the operations we take different amounts of polynomials. If the user chose normalisation or derivation, the application only asks for one polynomial, on the other hand, if the user picks addition or multiplication, we take an unlimited amount of polynomials, separated by commas. If the user picks the quitting option, the app will just close. In the case the user picked the derivation option, a prompt will appear, asking which variable they would like to derive the polynomial to. We tried to give as much information as possible to the user to facilitate the usage of the application.

3. Testing

(Example cases that allow the testing of the functionalities)
A list of commands to test each functionality

GHCI version

Running the GHCI version:

To run it please follow these steps:
1. Open the terminal on the same folder as "polyframework.hs"
2. Use the command "ghci"
3. When the ghci opens, use the command ":l polyframework.hs"

Parsing

build_polynomial ("10")
build_polynomial ("-10")
build_polynomial ("x")
build_polynomial ("-x")
build_polynomial ("xss")
build_polynomial ("x_1")
build_polynomial ("x^2")
build_polynomial ("10 * x")
build_polynomial ("x * 10")
build_polynomial ("10 * x^2")
build_polynomial ("x^2 * 10")
build_polynomial ("x^2 * 10 * 5")
build_polynomial ("10 * 5 * x * y")
build_polynomial ("10 * 5 * x^10 * y^10")
build_polynomial ("0*x^2 + 2*y + 5*z + y + 7*y^2")
build_polynomial ("-0*x^2 - 2*y - 5*z - y - 7*y^2")
build_polynomial ("-0*x^2 - 2*y - 5*z - y - 7*y^2")

Special Cases:
build_polynomial ("")

Normalization

normalizeStringPoly ("-0*x^2 - 2*y - 5*z - y - 7*y^2")
normalizeStringPoly ("6*y + x^2*y*x + 8*x^7")
normalizeStringPoly ("z*x*y + x*y*z + y*z*x")
normalizeStringPoly ("8*x*y^3 + 6*x^3 + 5*y*x*y^2")

Addition

Two Polynomials:
addStringPoly "6*y + x^2*y*x + 8*x^7" "4*x^4 + 8*y + 4*y*x^3"
addStringPoly "7*x*y + 4*x^3" "x + y + 6*x^3 + x*y"
addStringPoly "3*x + y" "7*x^2 + y^2 + x^3"

Many Polynomials:
addStringPolys "6*y + x^2*y*x + 8*x^7, x + y + 6*x^3 + x*y, 3*x + y"
addStringPolys "x, y, 5*z, 7*x*z*y, 4*x, 5*x*y^3"

Multiplication

Two Polynomials:
multiplyStringPoly "6*y + x^2*y*x + 8*x^7" "4*x^4 + 8*y + 4*y*x^3"
multiplyStringPoly "7*x*y + 4*x^3" "x + y + 6*x^3 + x*y"
multiplyStringPoly "3*x + y" "7*x^2 + y^2 + x^3"

Many Polynomials:
multiplyStringPolys "6*y + x^2*y*x + 8*x^7, x + y + 6*x^3 + x*y, 3*x + y"
addStringPolys "x, y, 5*z, 7*x*z*y, 4*x, 5*x*y^3"

Derivation

deriveStringPolynomial "x" "2 * x"
deriveStringPolynomial "y" "2 * x"
deriveStringPolynomial "x" "x^10"
deriveStringPolynomial "x" "2 * x"

GHC/.exe version

Running the GHC/.exe version:

To run it please follow this steps:
1. Open the terminal on the same folder as "polyfun.hs"
2. Use the command "ghc .\polyfun.hs"
3. Wait until the compilation is finished
4. Run the newly created "polyfun.exe" application

After running the application, just follow the prompts
(Please make sure to be **careful with your inputs**)

Normalization

Enter the number: 1
-0*x^2 - 2*y - 5*z - y - 7*y^2
Press Enter

Addition

Enter the number: 2
6*y + x^2*y*x + 8*x^7, x + y + 6*x^3 + x*y
Press Enter

Multiplication

Enter the number: 3
6*y + x^2*y*x + 8*x^7, x + y + 6*x^3 + x*y
Press Enter

Derivation

Enter the number: 4
6*y + x^2*y*x + 8*x^7
Press Enter
x
Press Enter