University of Canberra
Faculty of Information Sciences and Engineering

Computer Engineering 1 — 1997
Part 1 : Introductory Programming

Assignment 2
Due: Wednesday, 7th May 1997

Polynomial Package

Aims

This exercise builds on work begun in Practical Class 4, week 5. You are to develop *two separate scripts* that each implement a package of symbolic operations on polynomials. The two scripts will be based on different representations for the polynomial data structure. you should ensure that both your scripts are able to serve the *same client program* by making sure that the interfaces to the two scripts correspond exactly. Similarly, the observable behaviour of the main functions in the two scripts must be *identical*.

There are several aims of this assignment, beyond simply providing you with more opportunities to practice defining functions in Haskell. Unlike Assignment 1, you must *develop your own program design*, following the techniques discussed in lectures and in the textbook. Also, you will discover that the two different representations of polynomials will significantly affect the definitions that you write. When choosing a data representation you must be prepared to carefully evaluate the advantages and disadvantages of the various alternatives.

Outline

A *polynomial* in a single variable is a sum of *terms*, each having a real *coefficient* and a non-negative integer *exponent*, e.g.

$$8.3x^5 + 2.4x^2 - 3.7x + 18.7$$

$$4.7x^3 - 2.0x$$

The aim of the exercise is to produce two separate packages of functions which perform operations on polynomials such as addition, subtraction, differentiation with respect to x, and evaluation at a point (i.e. given x).

The two representations that you are to develop are:

1. a list of terms with non-zero coefficients
2. a list of coefficients (as in Practical Class 4)

Representation 1 — List of Terms
A polynomial can be represented as a list of terms, where a term is represented as a coefficient, exponent pair. In turn, coefficients can be represented by type `Float` and exponents can be represented by type `Int`.

```
> type Poly  = [Term]
> type Term  = (Coeff, Expon)
> type Coeff = Float
> type Expon = Int
```

Further, the representation will only include terms with *non-zero coefficients*, no two terms will have the *same exponent*, and the list of terms will be in *decreasing order* of exponents. For example, the polynomial

$$8.3x^5 + 2.4x^2 - 3.7x + 18.7$$

will be represented by

```
[(8.3,5), (2.4,2), (-3.7,1), (18.7,0)]
```

The zero polynomial will thus be represented by the empty list.

You may assume that the arguments to your functions are in normal form (i.e. no zero coefficients, no two terms with the same exponent, and the list of terms decreasing order of exponents). *However, you must ensure that the results of your functions are also in normal form.*

Representation 2 — List of Coefficients
A polynomial can also be represented as just a list of coefficients, such that the exponent of each term corresponds to the list index of each coefficient. As before, coefficients can be represented by type `Float` and exponents can be represented by type `Int`.

```
> type Coeff = Float
> type Expon = Int
> type Poly  = [Coeff]
```

This representation will include no zero coefficients beyond the last non-zero coefficient. (That is, the list will have *no trailling zeroes.*) For example, the polynomial

$$8.3x^5 + 2.4x^2 - 3.7x + 18.7$$

will be represented by

```
[18.7, -3.7, 2.4, 0.0, 0.0, 8.3]
```

As before, the zero polynomial will be represented by an empty list.

Again, you may assume that the arguments to your functions are in normal form (the last coefficient in the list is non-zero), but you must *ensure that your functions maintain normal form.*

The Main Functions
The main functions you are to develop are described below. As examples, consider the following two polynomials:

$$p1 = 8.3x^5 + 2.4x^2 - 3.7x + 18.7$$

$$p2 = 4.7x^3 - 2.0x$$

addition
The sum of two polynomials $p1 + p2$ is

$8.3x^5 + 4.7x^3 + 2.4x^2 - 5.7x + 18.7$

subtraction
The difference of two polynomials $p1 - p2$ is

$8.3x^5 - 4.7x^3 + 2.4x^2 - 1.7x + 18.$

evaluation at a point
The *value* of the polynomial *p1* at a point $x = 2.0$ is 286.5

derivative
The *derivative with respect to x* of polynomial *p1* is

$41.5x^4 + 4.8x - 3.7$

display
Write a function to display a polynomial in its standard readable form. For example, *p1* should be displayed as:

```
8.3x^5 +2.4x^2 -3.7x^1 +18.7
```

Notice that there are no terms with zero exponent printed, there is only one sign between each term (not ... `+ -3.7x^1` ..) and there is no leading plus sign. An optional improvement is to print `-3.7x` rather than `-3.7x^1` and `18.7` rather than `18.7x^0`.

Skeleton scripts are provided with type declarations for each of these functions. Of course, in the process of design, you will identify simpler functions that can be combined to build the main functions. (For example, in designing the polynomial display function, you may decide to define a function to display a single term.)

Remember that the results of each operation should be a polynomial in *normal form*. It is not sufficient to simply make the display function deal with zero coefficients and so on.

Testing

When you submit your assignment (see below) you must include a demonstration of the behaviour of your script on some sample test data. The test data is provided in the files `TermsExamples.lhs` and `CoeffsExamples.lhs` and `Test.lhs` contains some simple functions to run your functions on the test data. All these scripts are available in the directory `Scripts/Asst2`. Two skeleton scripts, `PolyTerms.lhs` and `PolyCoeffs.lhs` are also provided in that directory. Note that you must modify `Test.lhs` to import either the term modules or the coefficient modules. Inspect the script for more information. Of course, your own testing should be more thorough than just the tests provided.

Submission of Assignments

The standard arrangement for submission of this assignment is described below. Alternative arrangements may be discussed with your tutor.

By the due date, you should put listings of your two scripts and listings of two test output files, together with a floppy disk containing your two scripts in an envelope and place it in your tutor's mail box. Make sure your floppy disk is clear of any material that is not relevant to this assignment. You must not modify the `Test.lhs` script, other than the import statements as described above.

Rather than simply displaying the test results in a Hugs session, you will need to create files containing the test results. To do this, change `Test.lhs` to import the term scripts or the coefficient scripts respectively. Enter the following shell command (do not start Hugs from the button):

```
hugs Test.lhs << EOI > TermResults
testAll
EOI
```

This will produce a file called `TermResults`. Repeat the command but use a different file name (`CoeffResults` say) for the other representation. Do not modify the result files in any way.

Extensions

This is a particularly open-ended exercise and provides many opportunities for extensions if you are motivated and have the time. For example, you may wish to develop functions to perform multiplication. An interesting exercise is to develop functions that *normalise* polynomials in each representation. For example, for the first representation that means combining terms with the same exponent, eliminating terms with zero coefficient and sorting the terms into decreasing order of exponents. Also, you may wish to develop the package once again for a third, different, polynomial representation.