

Assignment 1: Haskell

Due: 11.59pm on Thursday 14th August, 2003.

This assignment falls due in the week following the two lab sessions that introduced Haskell programming. It is an assignment that is meant to consolidate the lessons learned there.

The submission will be electronic; it must be done using the marker-tool or the mark command:
`mark comp2600 ass1 filenames`

Late submissions will normally incur a penalty of ten percent per day and special arrangements that are required because of truly exceptional circumstances need to be negotiated before your deadline. In any case, solutions will be posted early in the following week, and no further submissions will be accepted.

We will attempt to run your submission and you will not have prior knowledge of the test data to be used. Therefore, you are advised to thoroughly exercise your functions.

1 Sorted Lists

Many common list operations can be written more efficiently if the list is known to be sorted into ascending order.

Searching There is a standard Haskell function `elem`, which determines whether a given value (`y`, say) occurs in a list of elements (`xs`, say) of the same type. If `y` does not actually appear in the list `xs`, the function still has to search right through the list to ascertain that fact.

If the list of elements is already sorted into ascending order, then searching can cease when the element is found, or when we know that the value of each remaining element is greater than the value of the element being searched for. Write a function `elemSort y xs` which assumes that the list `xs` is already sorted and which implements this more efficient searching technique.

Removing duplicates In the Haskell library module `List.hs`, there is a function `nub` which removes multiple occurrences of values from a list, leaving only a single occurrence.

As in the previous exercise, this is defined to work for lists in general. If the list is sorted, we can take advantage of the fact that multiple occurrences of values will be adjacent to each other in the list. Define a function `remDups xs` which removes duplicates from the list, leaving a single occurrence, but assumes that the list is sorted.

Submit your Haskell code for these two functions in a script called `SortedLists.hs`.

2 Polynomials

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 including addition, subtraction, and evaluation at a point.

The two representations that you are to develop are:

1. a list of terms with non-zero coefficients
2. a list of coefficients

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 $p1 + p2 = 8.3x^5 + 4.7x^3 + 2.4x^2 - 5.7x + 18.7$

subtraction

The difference of two polynomials $p1 - p2 = 8.3x^5 - 4.7x^3 + 2.4x^2 - 1.7x + 18$

evaluation at a point

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

Skeleton scripts are provided in /dept/dcs/comp2600/public/ass/ass1.

2.1 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.

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 is in decreasing order of exponents).

However, you must ensure that the results of your functions are also in normal form.

Define functions called `add`, `sub` and `eval` for polynomials represented as lists of terms (in normal form).

Submit your solution in a Haskell script called `TermPoly.hs`.

2.2 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 trailing 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, assume that the arguments to your functions are in normal form (the last coefficient in the list is non-zero).

Again, you must ensure that your functions maintain normal form.

Define functions called `add`, `sub` and `eval` for polynomials represented as lists of coefficients (in normal form).

Submit your solution in a Haskell script called `CoeffPoly.hs`.