University of Canberra
Faculty of Information Sciences and Engineering

Computer Engineering 1 — 1997
Part 1 : Introductory Programming

Practical Class 4 — Week 5

Aim

The aim of this class is to help you learn to define your own functions on lists, especially using recursion and list patterns.

Exercise 1 — Product of the elements of a list

There is a standard Haskell function `product` that multiplies together all the elements of a list of numbers, but I want you to define your own, say `myProduct`. It should be quite easy — look at `sumList` in Chapter 4.2 of the textbook for clues. The sample scripts from the textbook are available through the CE1 web page. Don't be tempted to copy the definition from the standard prelude, unless you think you can clearly explain the definition to your tutor and me!

Exercise 2 — Converting a string to upper case

Haskell has a pre-defined function `toUpper :: Char -> Char` which converts single lower case letters to upper case. Experiment with this function to find out exactly how it behaves (i.e. try it on a variety of characters).

Define a function that converts all the lower case letters in a *string* to upper case, and leaves the others unchanged. For example, `stringToUpper :: String -> String`.

Exercise 3 — Adding two polynomials

A polynomial in a single variable can be represented rather simply by a list of its coefficients. For example:

`[1, 7, 5, 2]`	represents	$2x^3 + 5x^2 + 7x + 1$
`[42, 2, 1]`	represents	$x^2 + 2x + 42$
`[-3, 0, 0, 0, 1]`	represents	$x^4 - 3$
`[0, -2, 0, 4]`	represents	$4x^3 - 2x$

Notice how the list index for each element corresponds to the exponent of the term.

Define a type synonym `Poly` for this representation. Two polynomials can be summed by adding the coefficients of corresponding terms. For example, the sum of $2x^3 + x^2 + 1$ and $3x^4 + 4x^2 - 7$ is $3x^4 + 2x^3 + 5x^2 - 6$.

Define a Haskell function `sumPoly :: Poly -> Poly -> Poly` that sums two polynomials that are represented as in the previous exercise. Take care with the case of polynomials of different degrees. For example:

```
? sumPoly [1, 7, 5, 2] [42, 2, 1]
[43, 9, 6, 2]
? sumPoly [-3, 0, 0, 0, 1] [1, 7, 5, 2]
[-2, 7, 5, 2, 1]
```

```
? evalPoly 4 (sumPoly [0, -2, 0, 4] [1, 7, 5, 2])
485
```

Again you should include appropriate comments, of course.

Exercise 4 — Evaluating a polynomial
We also want a Haskell function which, given a polynomial and a value for x, will calculate the value of the polynomial for that value of x. For example:

```
? evalPoly 3 [1, 7, 5, 2]
121
? evalPoly (-2) [0, -2, 0, 4]
-28
?
```

There are many ways to do this, but an identity that you may find helpful is the following:

$$a_n x^n + \cdots + a_2 x^2 + a_1 x + a_0 \quad = \quad a_0 + x(a_1 + x(a_2 + x(\cdots a_n)\cdots)$$

Make sure that your script includes appropriate comments.

Submission
You have now completed the main exercises. Make sure to show your solutions to your tutor for approval and assessment.

Extension
If you have time you might like to try the following exercise.

Exercise 5 — Master Mind
Back in the old days, we used to play a game called *Bulls and Cows* which was later marketed under the name of *Mastermind*. One player secretly picks a four letter word and the other player tries to guess it. If her guess is correct the game is over. Otherwise the first player says how many letters in her guess were in the secret word and in the correct position ('bulls'), and how many were in the word but in the wrong position ('cows'). For example, if the secret word was "meet" and the guess was "mute" the answer would be "1 bull and 2 cows."

Write a function `bulls :: String -> String -> Int` which counts the number of positions that two strings have the same character.

Write a function `cows :: String -> String -> Int` which counts how many characters two strings have in common that are not in the same position. That is, don't count characters that the `bulls` function would count.

For example:

```
? bulls "mute" "meet"
1
? cows "mute "meet"
2
```