Programming Paradigms
Practical session, Week 3

Department of Informatics
University of Beira Interior

Chapter 4 slides

1. Consider a function safetail that behaves in the same way as tail, except that safetail maps the empty list to the empty list, whereas tail gives an error in this case. Define safetail using:

   (a) a conditional expression;

   (b) guarded equations;

   (c) pattern matching.

   Hint: the library function

   ```
    null :: [a] => Bool
   ```

   can be used to test if a list is empty.

2. Give three possible definitions for the logical or operator (||) using pattern matching.

3. Redefine the following version of (&&) using conditionals rather than patterns:

   ```
   True && True = True
   _    && _    = False
   ```

4. Do the same for the following version:

   ```
   True  && b = b
   False && _ = False
   ```

Chapter 5 slides

1. Using a list comprehension, give an expression that calculates the sum $1^2 + 2^2 + ... + 100^2$ of the first one hundred integer squares. Give a second expression that calculates the sum of the first n integer squares.

2. A triple (x,y,z) of positive integers is called pythagorean if $x^2 + y^2 = z^2$. Using a list comprehension, define a function

   ```
   pyths :: Int => [(Int,Int,Int)]
   ```

   that maps an integer n to all such triples with components in [1..n]. For example:

   ```
   > pyths 5
   [(3,4,5),(4,3,5)]
   ```

3. A positive integer is perfect if it equals the sum of all of its factors, excluding the number itself. Using a list comprehension, define a function

   ```
   perfects :: Int => [Int]
   ```

   that returns the list of all perfect numbers up to a given limit. For example:

   ```
   > perfects 500
   [6,28,496]
   ```

4. The scalar product of two lists of integers xs and ys of length n is given by the sum of the products of the corresponding integers:

   $$\sum_{i=0}^{n-1} (xs_i \times ys_i)$$

   Using a list comprehension, define a function that returns the scalar product of two lists.

Chapter 6 slides

1. Without looking at the standard prelude, define the following library functions using recursion:

   (a) Decide if all logical values in a list are true:

   ```
   and :: [Bool] -> Bool
   ```

(b) Concatenate a list of lists:

```
concat :: [[a]] -> [a]
```

(c) Produce a list with n identical elements:

```
replicate :: Int a -> [a]
```

(d) Select the nth element of a list:

```
(!!) :: [a] -> Int a
```

(e) Decide if a value is an element of a list:

```
elem :: Eq a => a -> [a] -> Bool
```

(f) Define a recursive function

```
merge :: Ord a => [a] -> [a] -> [a]
```

that merges two sorted lists of values to give a single sorted list. For example:

```
> merge [2,5,6] [1,3,4]
[1,2,3,4,5,6]
```

2. Define a recursive function

```
msort :: Ord a => [a] -> [a]
```

that implements `merge sort`, which can be specified by the following two rules:

(a) Lists of length ≤ 1 are already sorted;

(b) Other lists can be sorted by sorting the two halves and merging the resulting lists.

3. Define a recursive function

```
euclid:: Int ->Int -> Int
```

that implements Euclid's algorithm for calculating the greatest common divisor of two non-negative integers:

(a) If the two numbers are equal, this number is the result;

(b) Otherwise, the smaller number is subtracted from the larger, and the same process is then repeated.

For example:

```
> euclid 6 27
3
```