# Programming Paradigms
# Written exam

Aalborg University

26 February 2025 13:00-16:00

**This is the cover page of the problem set. Please read it carefully before you begin!!!**

**What is this?** This problem set consists of 6 problems. The problem set is part of a zip archive together with a file called `solutions.hs`.

**How do I begin?** *Please do the following immediately!!* Unzip the zip archive – it is not enough to look within it; you *must* unzip it and save the problem set and `solutions.hs` locally on your computer. Otherwise, the answers that you write in `solutions.hs` will not be saved.

Please add your full name, AAU mail address and study number to the top of your local copy of `solutions.hs` saved to your computer where indicated in the file. If you experience problems with the file extension `.hs`, then rename the file while working and give it the file extension `.hs`, just before you submit it.

**Må jeg skrive på dansk?** Ja, det må du gerne. You can write your answers in Danish or in English.

**How and where should I write my solution?** Please write your answers by adding them to the local copy of `solutions.hs` on your computer. You must indicate in comments which problem your text concerns and which subproblem it concerns. All other text that is not runnable Haskell code (such as code that contains syntax errors or type errors) must also be written as comments.

Use the format as exemplified in the snippet shown below.

```
-- Problem 2.2

bingo = 17

-- The solution is to declare a variable called bingo with value 17.
```

**How should I submit my solution?** *Submit the local copy of `solutions.hs` that your solutions appear in and nothing else.* DO NOT SUBMIT A ZIP ARCHIVE OR A PDF FILE.

**Is Internet access allowed during the exam?** You are only allowed to use Internet access for getting this problem set and for submitting your answers to Digital Eksamen.

**What am I allowed to consult during the exam?** During the exam, you are only allowed to use the textbook *Programming in Haskell* by Graham Hutton, notes that you have written yourself and your local installation of the Haskell programming environment. *GitHub CoPilot, ChatGPT and other AI-based tools are not allowed.*

**What can I use for my code?** You are only allowed to use the Haskell `Prelude` for your Haskell code, unless the text of a specific problem specifically mentions that you should also use another specific module. Do not use any special GHCi directives.

**Is there anything else I must know?** Yes. Please read the text of each problem ***very carefully*** before trying to solve it. All the information you will need is in the problem text. If you have read all of the cover page so far, please write a comment at the top of your solution containing only the word cabbage. Please make sure that you understand what is being asked of you; it is a very good idea to read the text *more than once*.

## Problem 1 – 16 points

Below are four types. For each type, find an expression or definition that will have this particular type as its most general type *if you use type inference*. In each case, explain if the expression or definition is polymorphic and if it is, in which way it is polymorphic.

- Ord d => d -> d -> (Bool, Bool, d, d)
- Num b => (a, [a]) -> ([a], b)
- (a, b) -> p1 -> p2 -> p3 -> a
- [Char] -> ([Char], Bool)

## Problem 2 – 18 points

The goal of this problem is to define a function pairm that will take an element $x$ and a list $ys$ and will return a list of pairs where the first component of each pair is $x$ and the second component is an element of $ys$ that is different from $x$. The second components must appear in the same order as they appeared in $xs$.

As an example, pairm 1 [3,4,1,5,1,8] should give us [(1,3),(1,4),(1,5),(1,8)]. As another example, pairm "plip" ["plop","plip","plap","plup"] should give us [("plip","plop"),("plip","plap"),("plip","plup")].

1. Is pairm polymorphic? Give a precise explanation in a comment.
2. Define pairm using recursion.
3. Define pairm using list comprehension and with no use of recursion.
4. Define pairm using higher-order functions and with no use of recursion.

## Problem 3 – 16 points

The function runs takes a list and returns a list that tells us for each element $x$ in $xs$ how many consecutive appearences there are of $x$.

As an example, runs [1,2,2,2,3,4,5,5,1] should give us [1,3,1,1,2,1], as there is 1 occurrence of 1, followed by 3 occurrences of 2, followed by 1 occurrence of 3, followed by 1 occurrence of 4 and 2 occurrences of 5 and finally 1 more occurrence of 1.

1. Someone claimed that the type of runs that would be found using type inference would be [Int] ->[Int]. Explain in a comment why this is not correct and give the correct type of runs.
2. Define runs using recursion.

The function gcp takes two lists $xs$ and $ys$ and returns the list that is the greatest common prefix of $xs$ and $ys$.

As an example, gcp [1,2,3,5,6] [1,2,17,42] should return [1,2].

3. Define gcp using recursion.

## Problem 4 – 20 points

A *tree of whole numbers* is a tree in which nodes are labelled with integers and where each node can have zero or more (arbitrarily many) subtrees.

Figure 1 shows two trees of whole numbers, $t_1$ and $t_2$.

1. Define a datatype WNTree that describes the trees of whole numbers.
2. Show how one should represent $t_1$ and $t_2$ as values of type WNTree.
3. Define a function maxval that, when given a tree of whole numbers $t$, returns the maximal number found in $t$. In Figure 1 we should have that maxval $t_1$ gives us 484000 and that maxval $t_2$ gives us 8235. *Hint:* The function maximum in the Prelude is useful.

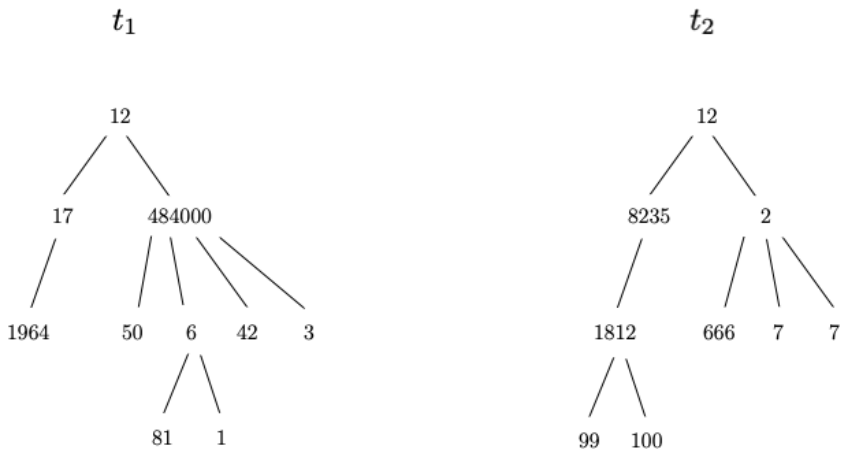


Figure 1: The trees of whole numbers $t_1$ and $t_2$.

## Problem 5 – 14 points

The function dublist takes a number $x$ and produces the infinite list of numbers

$$[x, 2 \cdot x, 2 \cdot 2 \cdot x, 2 \cdot 2 \cdot 2 \cdot x, \ldots]$$

As an example. 1.2 should give us the infinite list that begins with [1.2,2.4,4.8,9.6,19.2,38.4,...].

1. What should the type of dublist be?
2. Define dublist using recursion.
3. Define a version of dublist called dublist' that has the same type as dublist but is defined using list comprehension and without using recursion or the function dublist from the previous subproblem. *Hint:* In Haskell, we can write $4^2$ as 4^2.
4. Here is an expression.

   ```
   (\x y -> x + 3) 7 dublist
   ```

   Why does the evaluation of this expression succeed? Explain in a comment.

## Problem 6 – 16 points

Below is the declaration of a type Age and a declaration that makes it an instance of Functor. The idea is that data can be classified as old or new.

```
data Age a = New a | Old a deriving Show

instance Functor Age where
  fmap f (New x) = New (f x)
  fmap f (Old x) = Old (f x)
```

1. Extend the above piece of code with an instance declaration such that Age becomes an applicative functor also. An element that we wrap in this type becomes classified as new, if we apply a new function to it. Applying a function that is classified as old will always give us a value, that becomes classified as old.
2. Extend the above piece of code with an instance declaration such that Age also becomes a monad.
3. Use a do-block in the Age-monad to define a function

   ```
   rejuvenate :: Ord b => Age b -> Age b -> Age b
   ```

   which, given two values of type Age b where b is a type in Ord gives us the minimum value and re-classifies this minimum value to new.

   rejuvenate (New 4) (New 8) should give us New 4 and rejuvenate (Old "ab") (Old "aa") should give us New "aa".